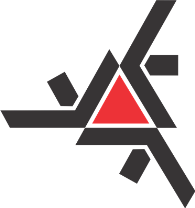
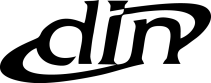
**UNIVERSIDADE ESTADUAL DE MARINGÁ**

**DEPARTAMENTO DE INFORMÁTICA**

**Curso:** Ciência da Computação

**Disciplina:** 6903 - Modelagem e Otimização Algorítmica

**Docente:** Ademir Aparecido Constantino

| **Discentes** | **RA** |
| --- | --- |
| Gabriel Henrique Costanzi | 102573 |
| Thiago Yuji Yoshimura Yamamoto | 112649 |

**Avaliação 3**

**Implementação de Algoritmos Genéticos**

# 

**Sumário**

[**1) Introdução**](#_rdv0ppp1xol7) **3**

[**2) Descrição do Problema**](#_cdhdeo3h76or) **3**

[**3) Descrição do Algoritmo**](#_xqh9btpkary) **3**

[3.1) Leitura dos dados de entrada](#_w61mrgy9oh4) 3

[3.2) Armazenamento dos dados de entrada](#_l0lfysixlgwp) 4

[3.3) Estrutura para representação de uma solução](#_1gk713lug5th) 5

[3.4) Estrutura para armazenamento da população de soluções](#_dfmzbopyyj3y) 5

[3.4.1) Armazenamento da população final](#_wde1v6bzxvxd) 5

[3.4.2) Armazenamento dos novos indivíduos gerados na geração](#_wx7127s0emm9) 5

[3.5) Algoritmo Genético](#_34e3qo371iky) 5

[3.5.1) Critérios de parada do algoritmo genético](#_7pm7v54a5co) 5

[3.5.2) Parâmetros do Algoritmo Genético](#_uqb51a5qtqtg) 5

[3.5.3) Etapa Inicial - Inicialização da População](#_5wyulkiexh86) 6

[3.5.4) Etapa 1 - Aptidão](#_k50rnru4cqim) 6

[3.5.6) Etapa 3 - Cruzamento](#_dx8z8jzcp2js) 6

[3.5.6.1) Operador de Cruzamento - OX1](#_p4af1ywc1nbb) 7

[3.5.8) Etapa 5 - Busca Local](#_758cmfqnx0bp) 7

[3.5.9) Etapa 5 - Atualização](#_ydg9k4cyrsh2) 8

[**3) Resultados**](#_h4hadeal9ekg) **9**

[3.1) PCV - pr1002.tsp](#_pq0jmi40pc9h) 11

[3.2) PCV - fnl4461.tsp](#_7nimh0yge8rg) 14

[3.3) PCV - pla7397.tsp](#_h6a6ux8hswka) 17

[3.4) PCV - brd14051.tsp](#_ux14a6t7syy9) 20

[3.5) PCV - d15112.tsp](#_2s1w7cntsm7k) 23

[3.6) PCV - d18512.tsp](#_qut8rdwhw8hd) 26

[3.7) PCV - pla33810.tsp](#_2qd2w3bku5m4) 29

[3.8) PCV - pla85900.tsp](#_hsevxf43mzlh) 32

[**4) Conclusão dos Resultados**](#_9mjusong4s) **35**

# 

# 1) Introdução

O objetivo deste trabalho é pesquisar e praticar a implementação de um algoritmo baseado na meta-heurística Algoritmos Genéticos combinado com Busca Local, aplicadas no problema do Caixeiro Viajante. Para isso, foi implementado dois operadores de cruzamento diferentes (OX1 e OX2) em um conjunto de instâncias, para que possamos estudar os resultados e a influência dos parâmetros utilizados.

Os algoritmos genéticos são métodos computacionais que têm como objetivo encontrar soluções satisfatórias para problemas de busca e otimização, no qual são inspirados nos mecanismos de evolução natural e recombinação genética (princípio Darwiniano de reprodução e sobrevivência dos mais aptos). Estes mecanismos são imitados a partir de um processo evolucionário que envolve avaliação, seleção, recombinação sexual (crossover através de cromossomos artificiais) e mutação, para que no fim de vários ciclos (chamado de gerações) a população deverá conter os indivíduos mais aptos (o que melhor atende a função objetiva). Para o nosso trabalho, o processo de busca local foi adicionado para aumentar a eficiência do algoritmo.

Os algoritmos de busca local são métodos que compõe para cada solução um conjunto de soluções vizinhas (soluções com características muito próximas), no qual são percorridas até que a solução corrente é um ótimo local em relação à vizinhança adotada.

# 2) Descrição do Problema

O Problema do Caixeiro Viajante (PCV) tem como problema definir um circuito com o menor custo de n cidades, que se inicia e encerra sua viagem na primeira cidade, sendo que deve-se visitá-los apenas uma única vez, não importando a ordem e que de cada uma delas pode-se ir diretamente a qualquer outra.

# 3) Descrição do Algoritmo

Para a melhor compreensão dos algoritmos implementados neste trabalho, a descrição de cada algoritmo será feita no formato de pseudo-código, mas vale ressaltar que alguns conceitos da linguagem Java serão apresentados (ex.: ArrayList, HashMap, Classes e Objetos).

## 3.1) Leitura dos dados de entrada

O arquivo contendo os dados de entrada deverá estar com extensão “.tsp” e o seu conteúdo como no seguinte exemplo:

| NAME : a8  COMMENT : exemplo de arquivo  TYPE : TSP  DIMENSION: 8  EDGE\_WEIGHT\_TYPE : 8 vertices  NODE\_COORD\_SECTION  1 288 149  2 288 129  3 270 133  4 256 141  5 256 157  6 246 157  7 236 169  8 228 169  EOF |
| --- |

Para fazer a leitura dos dados de entrada, pula-se as 6 primeiras linhas e armazene cada valor em os valores do identificador e das coordenadas, como no pseudo-código a seguir (considere a variável ponteiro como sua posição no arquivo de entrada):

| **while(**ponteiro has not Number**):**  mova o ponteiro para a próxima linha  **while(**ponteiro has Number**):**  identificador = ponteiro.nextNumber  x = ponteiro.nextNumber  y = ponteiro.nextNumber  vertice = Vertice(identificador, x, y)  listVertice.add(vertice) |
| --- |

## 3.2) Armazenamento dos dados de entrada

Os dados de entrada foram armazenados em uma ArrayList de objetos da classe Vertice.

Para cada vértice presente no arquivo de entrada, cria-se uma instância da classe Vertice atribuindo os valores de identificador e suas coordenadas, por fim adiciona-se no final da lista de vértices pré criada.

## 3.3) Estrutura para representação de uma solução

A classe Caminho foi implementada para representar uma das soluções para o problema, no qual tem como atributo:

* ArrayList de vértices (classe Vertice): armazenamento do caminho sequencial de 0 até última posição;
* Fitness: custo do peso do caminho total.

## 3.4) Estrutura para armazenamento da população de soluções

### 3.4.1) Armazenamento da população final

Como a população consiste de várias soluções e as soluções são os caminhos do problema, armazenamos o conteúdo da população dentro de um ArrayList de objetos da classe Caminho.

### 3.4.2) Armazenamento dos novos indivíduos gerados na geração

Os novos indivíduos gerados em uma determinada geração são armazenados em um ArrayList de Caminho separados da população final, porém o seu princípio de funcionamento é o mesmo.

## 3.5) Algoritmo Genético

### 3.5.1) Critérios de parada do algoritmo genético

Os critérios de parada considerados no algoritmo genético deste trabalho, foram:

* Quantidade de gerações máxima: foi definido um valor inteiro constante, para determinar o número máximo de gerações;
* Tempo de processamento: dependendo do tamanho das instâncias utilizadas, o tempo de processamento do algoritmo genético será muito elevado, necessitando definir um tempo máximo de processamento;
* Estagnação: para poupar tempo de execução, definimos um valor constante inteiro para representar o número máximo de gerações que não há melhoramento.

### 3.5.2) Parâmetros do Algoritmo Genético

* tamanho\_populacao: Representa o tamanho da população que será considerada ao executar o algoritmo.
* posicoes\_cruzamento: Representa a quantidade de posições aleatórias que serão utilizadas no algoritmo OX2.
* busca\_local\_profundidade: Representa a quantidade de melhorias que será feita para cada execução do algoritmo de busca local.
* taxa\_busca\_local: Representa a porcentagem de indivíduos da nova população que serão melhorados com busca local.
* taxa\_cruzamento: Representa a porcentagem de indivíduos da população que vão participar do cruzamento.
* taxa\_mutacao: Representa a probabilidade de indivíduos da nova população sofrerem mutação.
* taxa\_sobrevivencia: Representa a porcentagem de indivíduos da população corrente que vão sobreviver para a próxima geração.

### 3.5.3) Etapa Inicial - Inicialização da População

A etapa de inicialização da população é realizada através da utilização de um algoritmo de construção, que será executado n vezes, onde n é o tamanho da população, escolhendo o vértice inicial de forma aleatória, proporcionando uma boa diversidade na população inicial.

Neste trabalho, o algoritmo de construção utilizado foi o algoritmo do vizinho mais próximo.

### 3.5.4) Etapa 1 - Aptidão

A etapa de aptidão do algoritmo genético é feita pela medição da aptidão da função objetiva, neste caso, para o problema do caixeiro viajante a minimização do peso do caminho.

Para implementar esta etapa, utilizamos a classe Collections e o método Collections.sort presente na biblioteca linguagem Java. Desta forma ordenamos a população de acordo com o fitness de cada solução, sendo que a primeira posição será ocupada pelo indivíduo mais apto e a última pelo menos apto.

3.5.5) Etapa 2 - Seleção

A etapa de seleção do algoritmo genético é o processo de selecionamento dos indivíduos para o cruzamento (crossover), no qual é baseada em sua aptidão (ou seja, os mais aptos têm maior probabilidade de serem escolhidos para reprodução). Desta forma, o critério de selecionamento utilizado foi o Método da Roleta.

No Método da Roleta, cada indivíduo da população é representado na roleta de forma proporcional ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão é dada uma porção maior da roleta, enquanto aos de aptidão mais baixa é dada uma porção relativamente menor da roleta.

### 3.5.6) Etapa 3 - Cruzamento

A etapa de cruzamento do algoritmo genético é realizada por meio da combinação genética dos cromossomos pais selecionados na etapa anterior para a formação de cromossomos filhos. Os operadores de cruzamento vão definir de que forma a combinação genética vai ocorrer.

Os algoritmos operadores de cruzamento foram baseados no pseudo-código disponibilizado pelo docente deste trabalho.

#### 3.5.6.1) Operador de Cruzamento - OX1

O operador de cruzamento OX1 funciona por meio da seleção aleatória de dois pontos de corte (i, j), onde j é maior que i. A sequência de cidades entre esses dois pontos de corte é retirada do pai 1 e colocado no filho 1, e o mesmo ocorre para o pai 2 e filho 2. Após isso, os espaços restantes são preenchidos pelo outro pai colocando as cidade que ainda não estão contidas no filho na mesma sequência em que aparecem no pai.

Para uma compreensão melhor do operados OX1 abaixo será demonstrado os passos para cada solução pai:

* Selecione aleatoriamente duas posições de corte;
* Copie o segmento entre os pontos de corte do pai 1 para o filho 1 e faça o mesmo procedimento para o pai 2 e filho 2;
* Para as posições que ainda faltam, copie as cidades do outro pai a partir do ponto de corte posterior na mesma ordem em que elas aparecem no pai.

3.5.6.2) Operador de Cruzamento - OX2

O operador de cruzamento OX2 funciona por meio da seleção de várias posições do cromossomo pai, e a partir dos vértices contidas nessas posição, elas deverão ser sobrepostas no outro progenitor com a mesma ordem de precedência do progenitor corrente.

Para uma compreensão melhor do operador OX2 abaixo será demonstrado os passos para cada solução pai:

* Escolha n posições do outro cromossomo pai;
* Crie uma solução intermediária copiando as cidades do pai corrente com exceção das cidades que estão contidas nas n posições do outro pai;
* Copie as cidades do outro pai nas posições faltantes do pai corrente, conforme a ordem de incidência do outro pai.

3.5.7) Etapa 4 - Mutação

A etapa de mutação do algoritmo genético é um operador exploratório que tem como objetivo aumentar a diversidade na população, para isso troca-se o conteúdo de uma posição do cromossomo.

Neste trabalho, a forma de se efetuar a mutação foi feita a partir da troca de duas posições do cromossomo da solução escolhida aleatoriamente. Esta troca gerará novos indivíduos para a população da geração corrente.

### 3.5.8) Etapa 5 - Busca Local

A etapa de busca local do algoritmo genético é feita a partir da aplicação do algoritmo de busca local 2-OPT com uma quantidade de melhorias específica, em uma porcentagem da nova população gerada no decorrer das etapas anteriores. Esta aplicação gerará novos indivíduos.

### 3.5.9) Etapa 5 - Atualização

A etapa de atualização do algoritmo genético é feita a partir da junção entre a população corrente e a nova população (contendo todos os filhos gerados a partir do decorrer das etapas anteriores). Neste trabalho, a técnica utilizada como critério de substituição dos novos indivíduos foi o elitismo, no qual definimos uma porcentagem de indivíduos que se manterão para a próxima geração, substituindo o restante da população.

# 3) Resultados

Os resultados da tabela a seguir demonstram os valores obtidos para o algoritmo genético desenvolvido neste trabalho e algoritmos heurísticos desenvolvidos no trabalho anterior a este, a fim de analisá-los na seção Conclusão dos Resultados.

Tabela 1 - Resultados do algoritmo genético

| **Caso** | **MS** | **OX1** | **GAP OX1** | **OX2** | **GAP OX2** |
| --- | --- | --- | --- | --- | --- |
| **pr1002** | 259045 | 268229 | 3,54 | 265697 | 2,56 |
| **fnl4461** | 182566 | 193573 | 6,02 | 192604 | 5,49 |
| **pla7397** | 23260728 | 24979126 | 7,38 | 24667484 | 6,04 |
| **brd14051** | 469385 | 499022 | 6,31 | 498505 | 6,20 |
| **d15112** | 1573084 | 1680397 | 6,82 | 1685209 | 7,12 |
| **d18512** | 645238 | 686370 | 6,37 | 684421 | 6,07 |
| **pla33810** | 66048945 | 70673985 | 7,00 | 70707196 | 7,05 |
| **pla85900** | 142382641 | 156420331 | 9,85 | 153850009 | 8,05 |

Tabela 2 - Resultados obtidos para os algoritmos do trabalho 1

| **Caso** | **MS** | **2-OPT** | **GAP 2-OPT%** | **3-OPT** | **GAP 3-OPT%** |
| --- | --- | --- | --- | --- | --- |
| **pr1002** | 259045 | 283036 | 9,26 | 279436 | 7,87 |
| **fnl4461** | 182566 | 194413 | 6,48 | 194965 | 6,79 |
| **pla7397** | 23260728 | 25098869 | 7,90 | 24937101 | 7,20 |
| **brd14051** | 469385 | 501661 | 6,87 | 500542 | 6,63 |
| **d15112** | 1573084 | 1691189 | 7,50 | 1687790 | 7,29 |
| **d18512** | 645238 | 687867 | 6,60 | 685777 | 6,28 |
| **pla33810** | 66048945 | 71067031 | 7,59 | 71123446 | 7,68 |
| **pla85900** | 142382641 | 151378176 | 6,31 | 151349267 | 6,29 |

Vale ressaltar que o algoritmo foi implementado na linguagem Java, a IDE Visual Studio Code e executado em uma máquina com as especificações:

* Processador: Ryzen 5 5600x
* 16GB RAM

Caso queira analisar os dados fornecidos para o desenvolvimento dos gráficos desta seção, acesse o link:

<https://docs.google.com/spreadsheets/d/1hlviOSaGTYRWoKRHSscKUQE4NYj8FxiObA6rtj2HZOw/edit?usp=sharing>

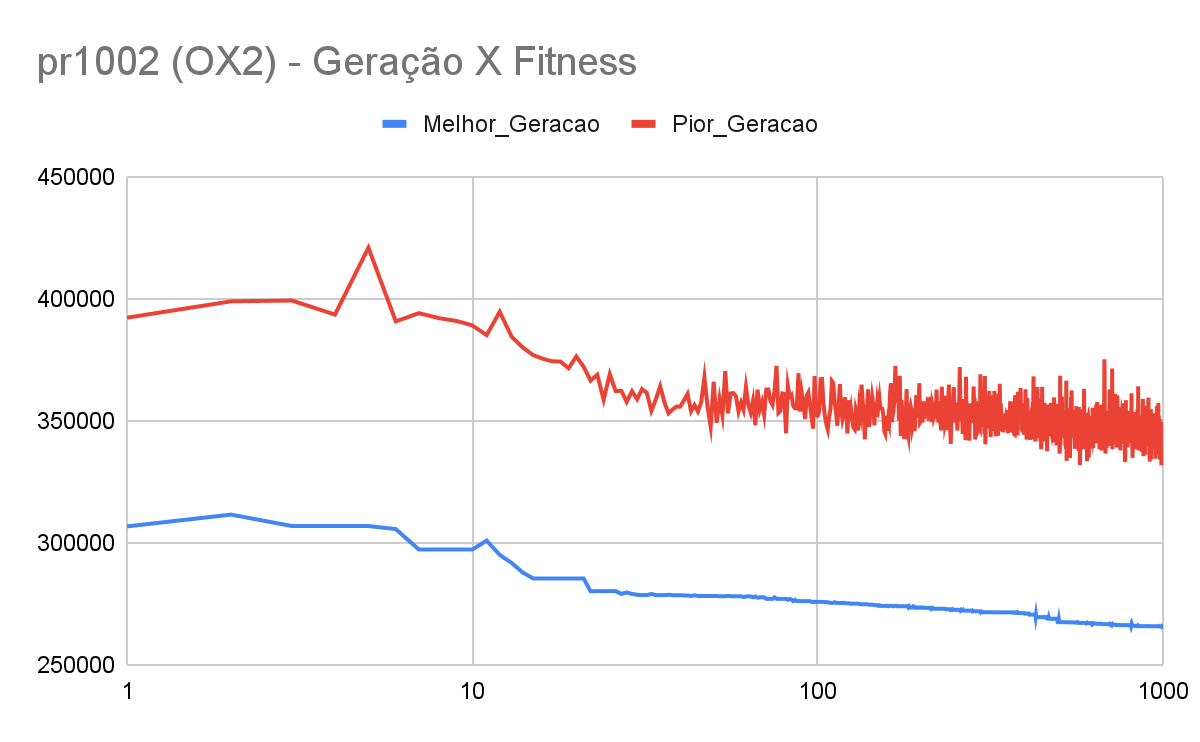
## 3.1) PCV - pr1002.tsp

Tabela 3 - Informações do critério de parada (pr1002)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 813 | 459 | ESTAGNACAO |
| **OX2** | 1000 | 240 | MAX\_GERACAO |

Tabela 4 - Parâmetros utilizados(pr1002)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 100 | 100 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 50 | 50 |
| **Taxa de busca local** | 20% | 10% |
| **Taxa de cruzamento** | 80% | 80% |
| **Taxa de mutação** | 5% | 10% |
| **Taxa de sobrevivência** | 40% | 30% |

Figura 1 - Gráfico do melhor e pior fitness de cada geração do pr1002 - OX2

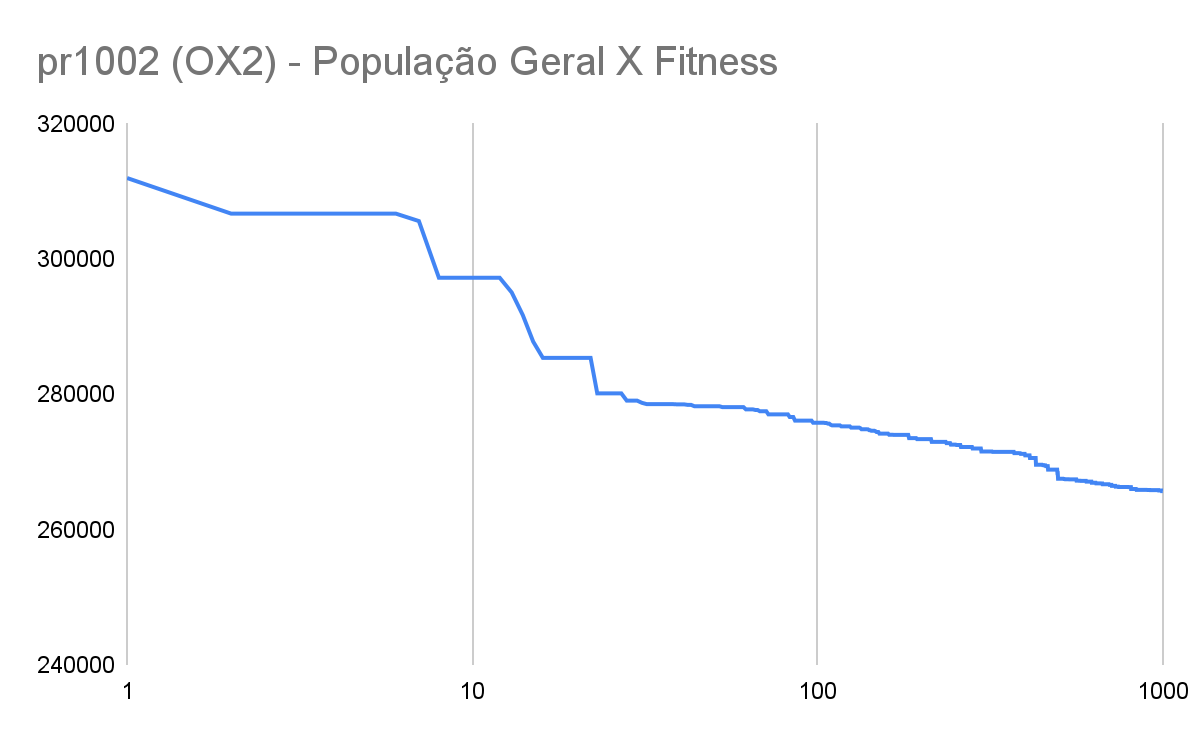
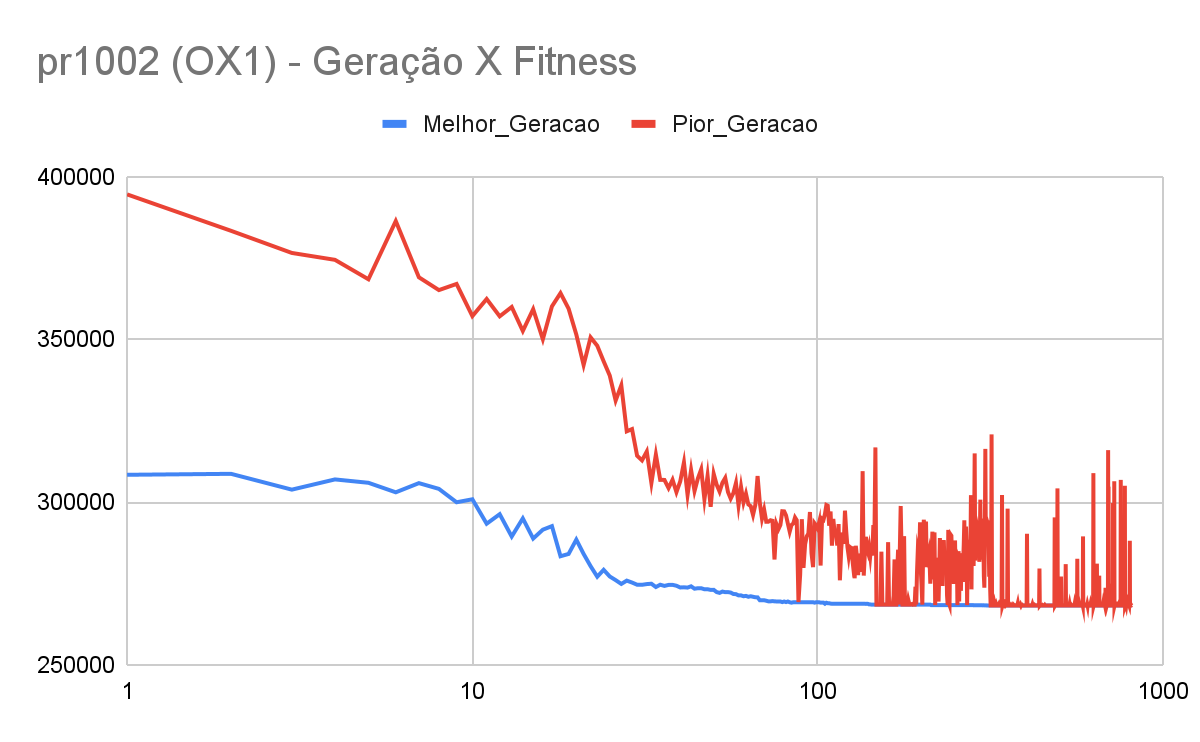
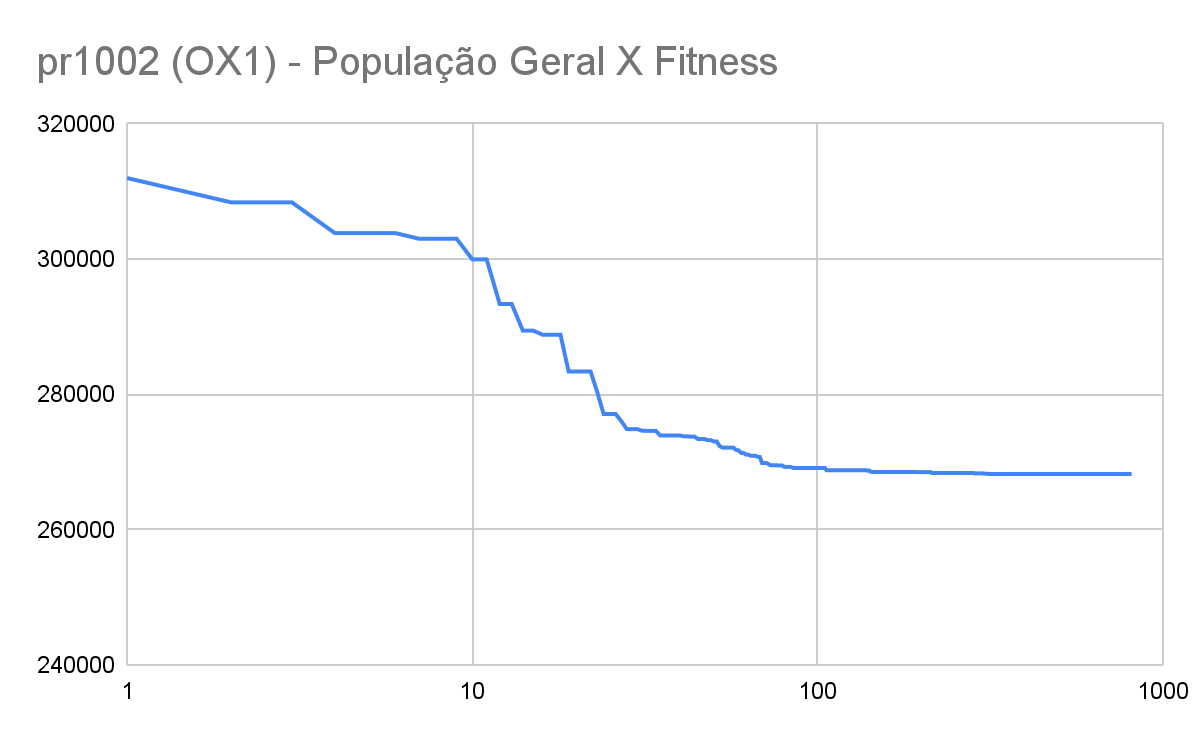


Figura 2 - Gráfico do melhor fitness da população geral do pr1002 - OX2

****Figura 3 - Gráfico da melhoria dos pesos da população geral do pr1002 - OX1

****Figura 4 - Gráfico do melhor fitness da população geral do pr1002 - OX1

## 3.2) PCV - fnl4461.tsp

Tabela 5 - Informações do critério de parada (fnl4461)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 455 | 1010 | ESTAGNACAO |
| **OX2** | 1000 | 833 | MAX\_GERACAO |

Tabela 6 - Parâmetros utilizados(fnl4461)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 100 | 100 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 50 | 50 |
| **Taxa de busca local** | 10% | 10% |
| **Taxa de cruzamento** | 90% | 90% |
| **Taxa de mutação** | 10% | 5% |
| **Taxa de sobrevivência** | 60% | 40% |

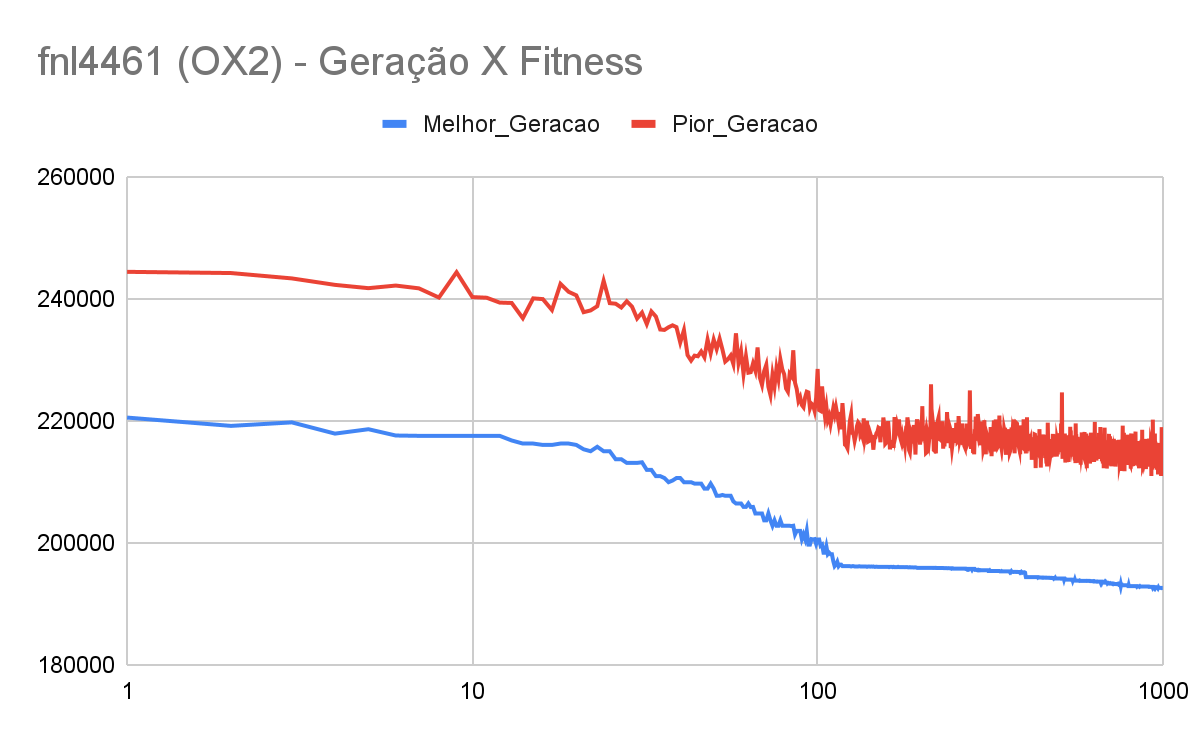
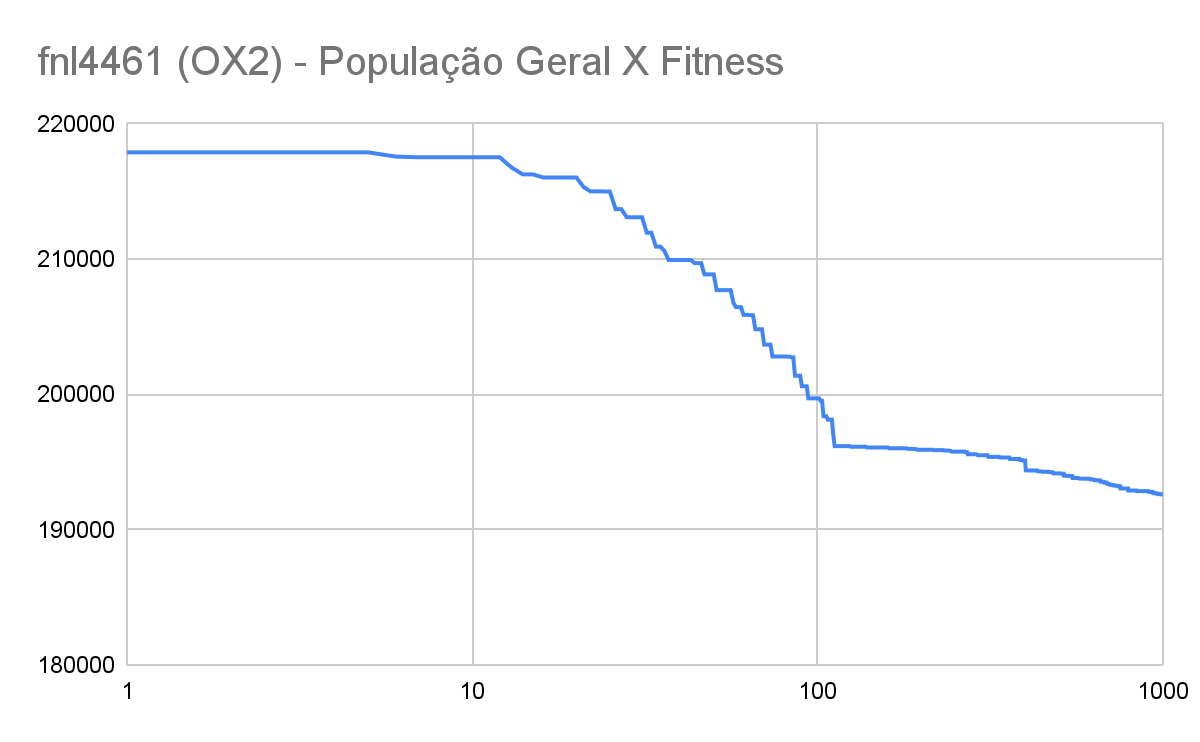
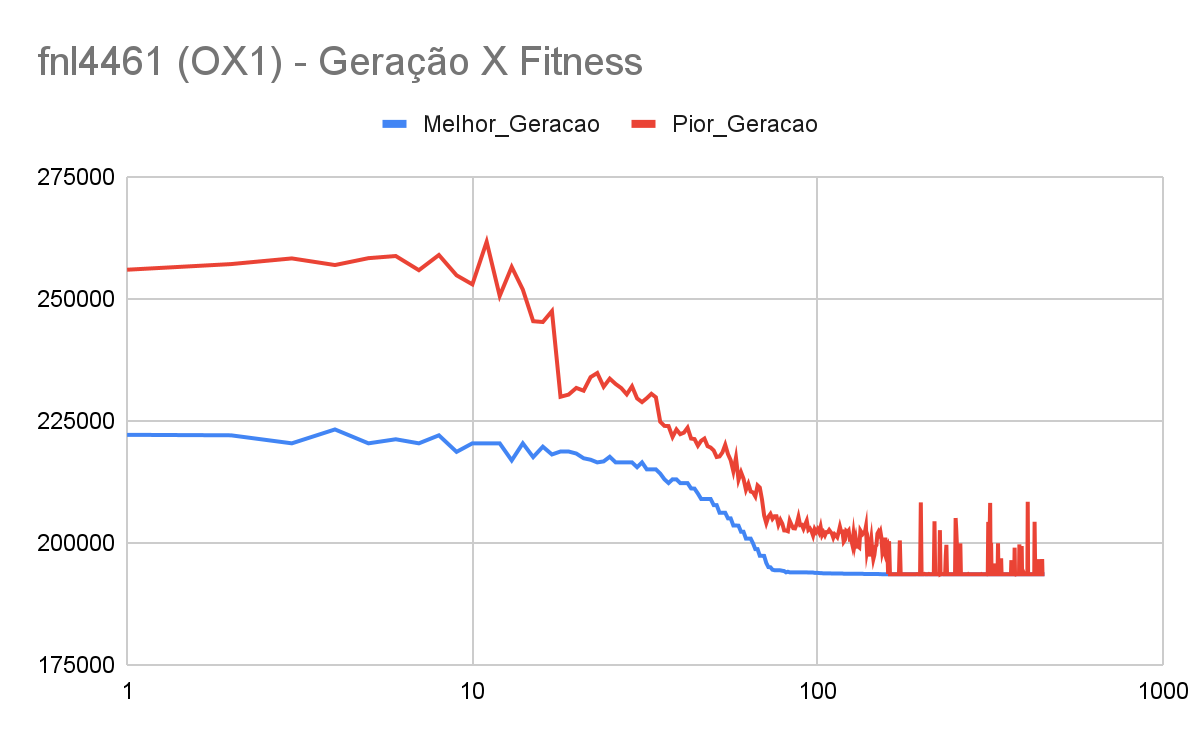
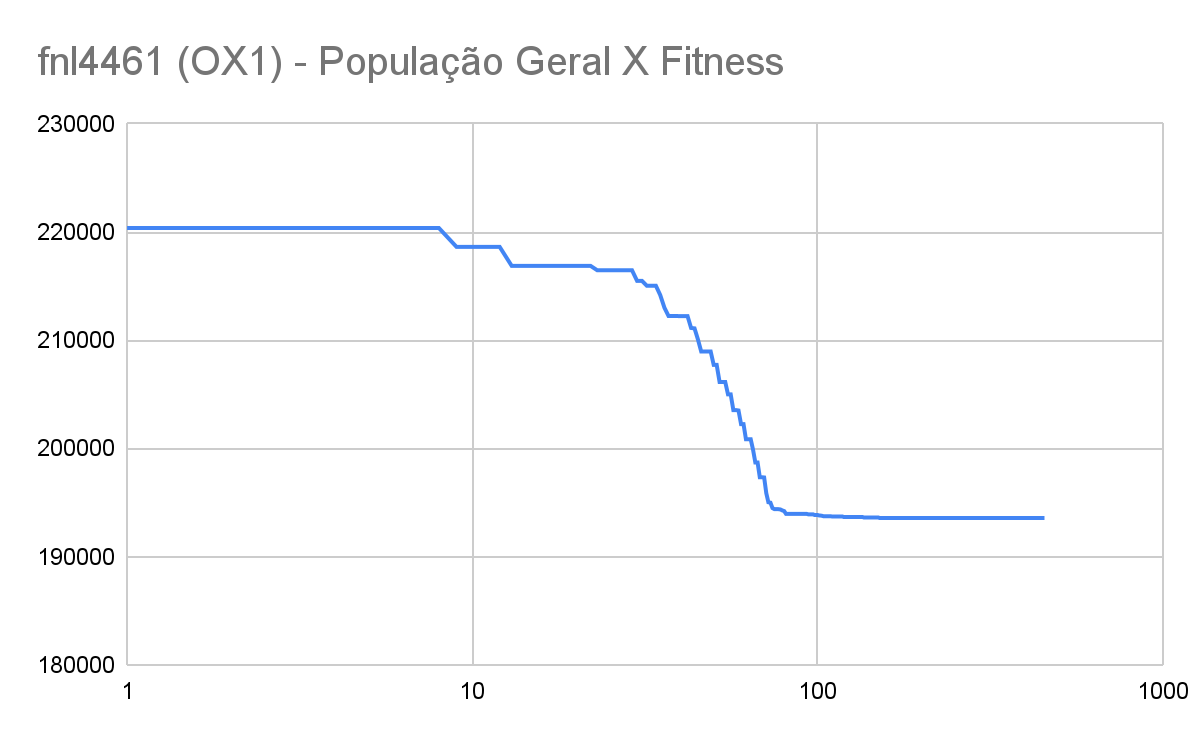
****

Figura 5 - Gráfico do melhor e pior fitness de cada geração do fnl4461 - OX2

****Figura 6 - Gráfico do melhor fitness da população geral do fnl4461 - OX2

****Figura 7 - Gráfico do melhor e pior fitness de cada geração do fnl4461 - OX1

****Figura 8 - Gráfico do melhor fitness da população geral do fnl4461 - OX1

## 3.3) PCV - pla7397.tsp

Tabela 7 - Informações do critério de parada (pla7397)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 631 | 9000 | TEMPO\_ESGOTADO |
| **OX2** | 1000 | 4739 | MAX\_GERACAO |

Tabela 8 - Parâmetros utilizados(pla7397)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 100 | 100 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 50 | 50 |
| **Taxa de busca local** | 10% | 10% |
| **Taxa de cruzamento** | 80% | 80% |
| **Taxa de mutação** | 5% | 10% |
| **Taxa de sobrevivência** | 40% | 30% |

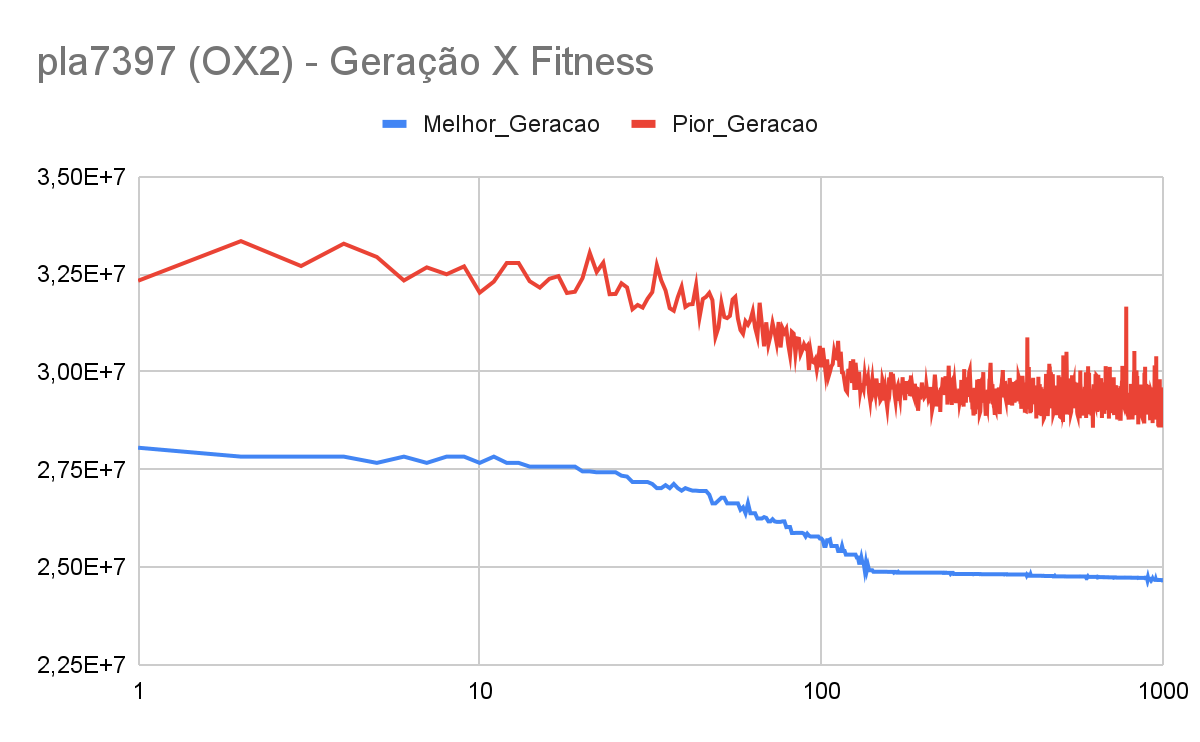
****

Figura 9 - Gráfico do melhor e pior fitness de cada geração do pla7397 - OX2

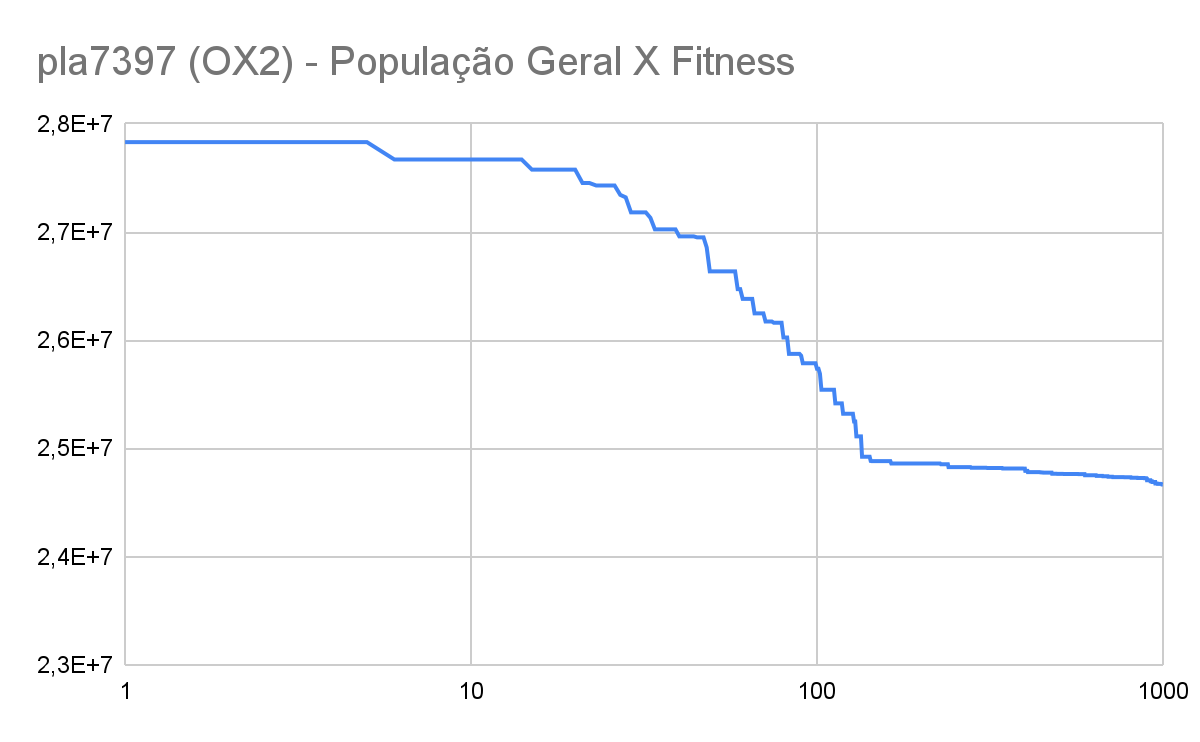
****

Figura 10 - Gráfico do melhor fitness da população geral do pla7397 - OX2

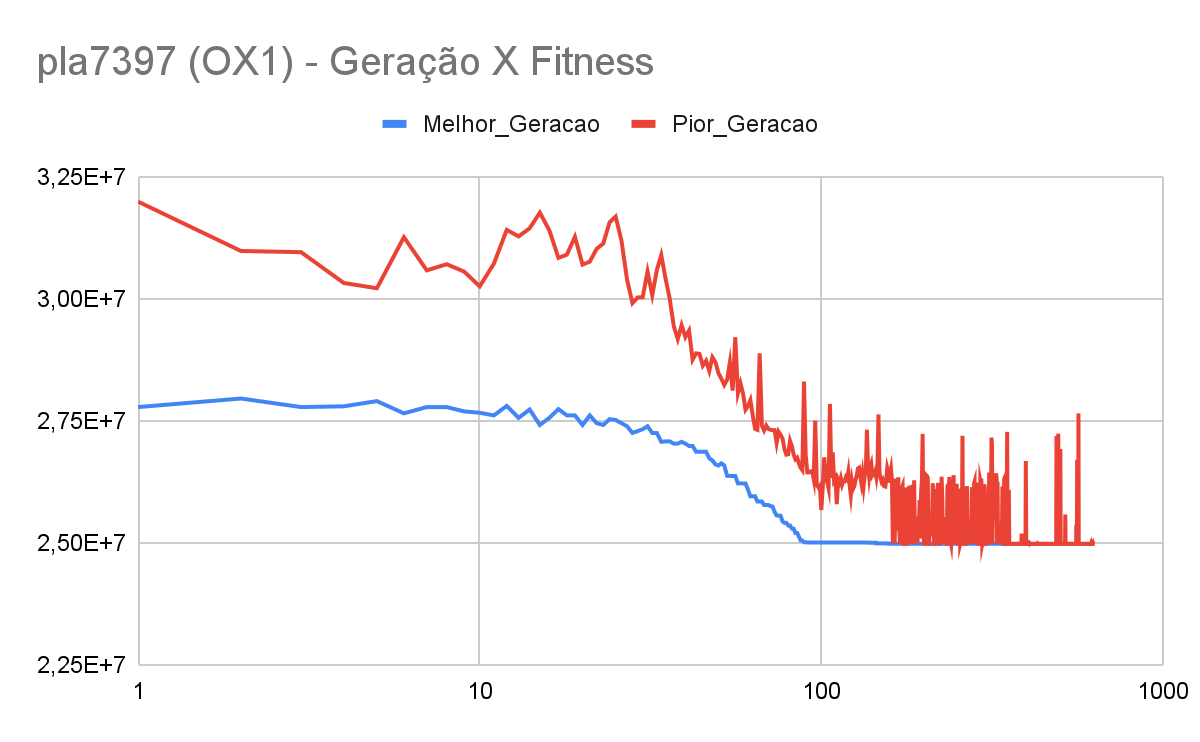
****

Figura 11 - Gráfico do melhor e pior fitness de cada geração do pla7397 - OX1

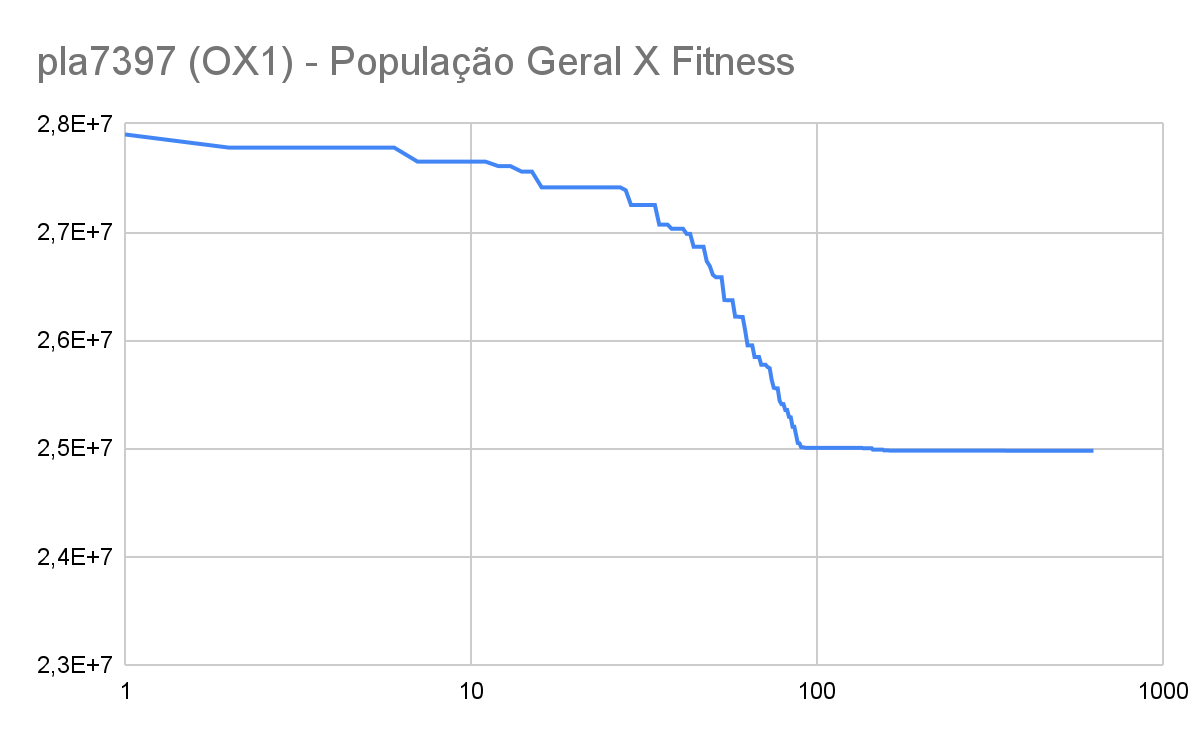
****

Figura 12 - Gráfico do melhor fitness da população geral do pla7397 - OX1

## 3.4) PCV - brd14051.tsp

Tabela 9 - Informações do critério de parada (brd14051)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 499 | 5347 | ESTAGNACAO |
| **OX2** | 1000 | 6397 | MAX\_GERACAO |

Tabela 10 - Parâmetros utilizados(brd14051)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 50 | 100 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 50 | 50 |
| **Taxa de busca local** | 10% | 10% |
| **Taxa de cruzamento** | 90% | 90% |
| **Taxa de mutação** | 10% | 5% |
| **Taxa de sobrevivência** | 60% | 40% |

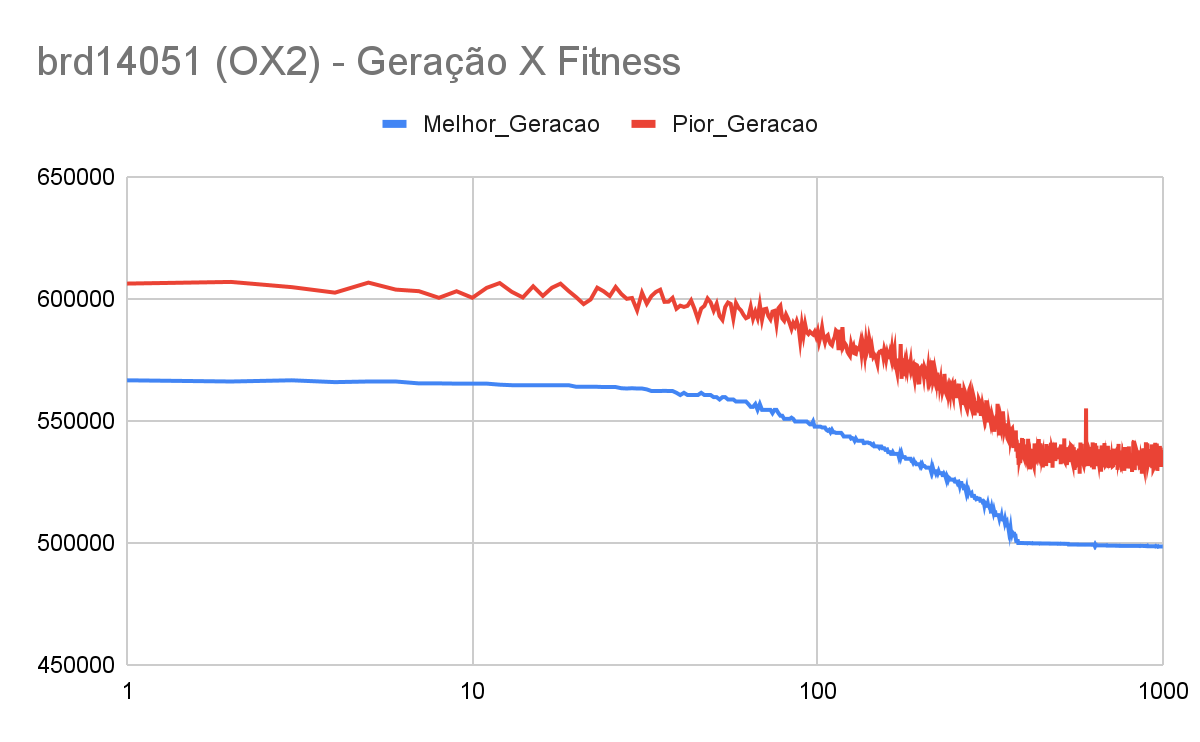


Figura 13 - Gráfico do melhor e pior fitness de cada geração do brd14051- OX2

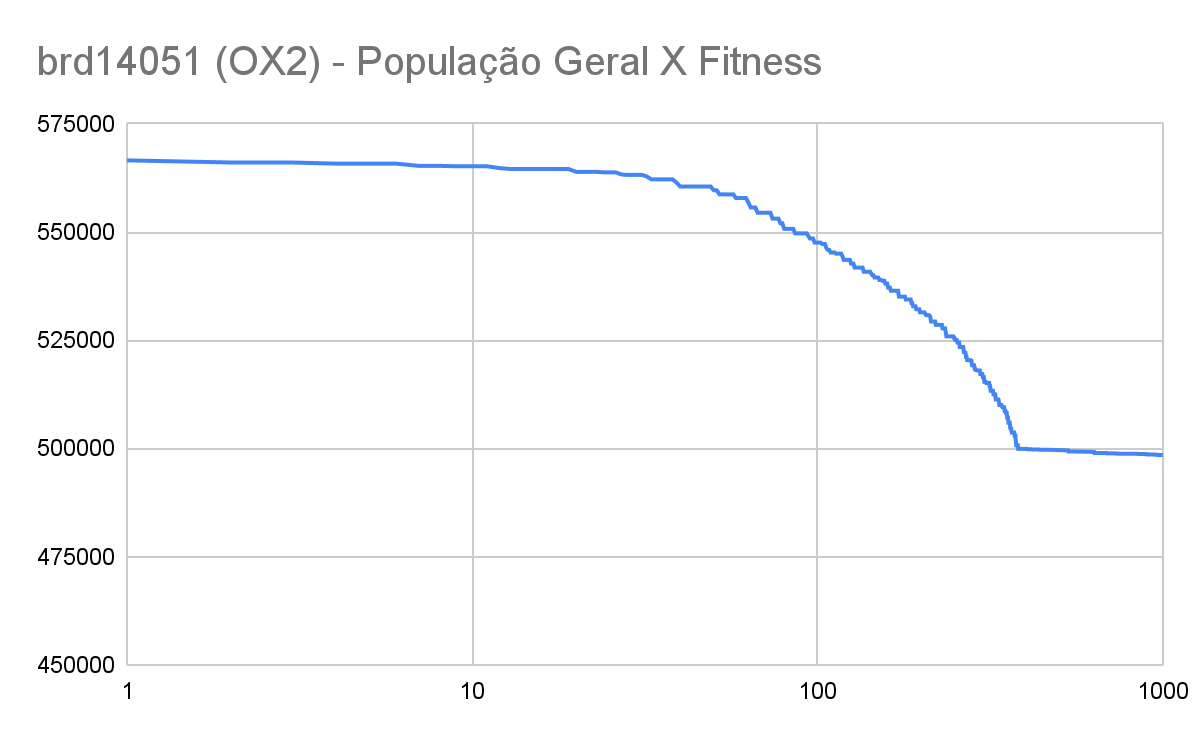


Figura 14 - Gráfico do melhor fitness da população geral do brd14051 - OX2

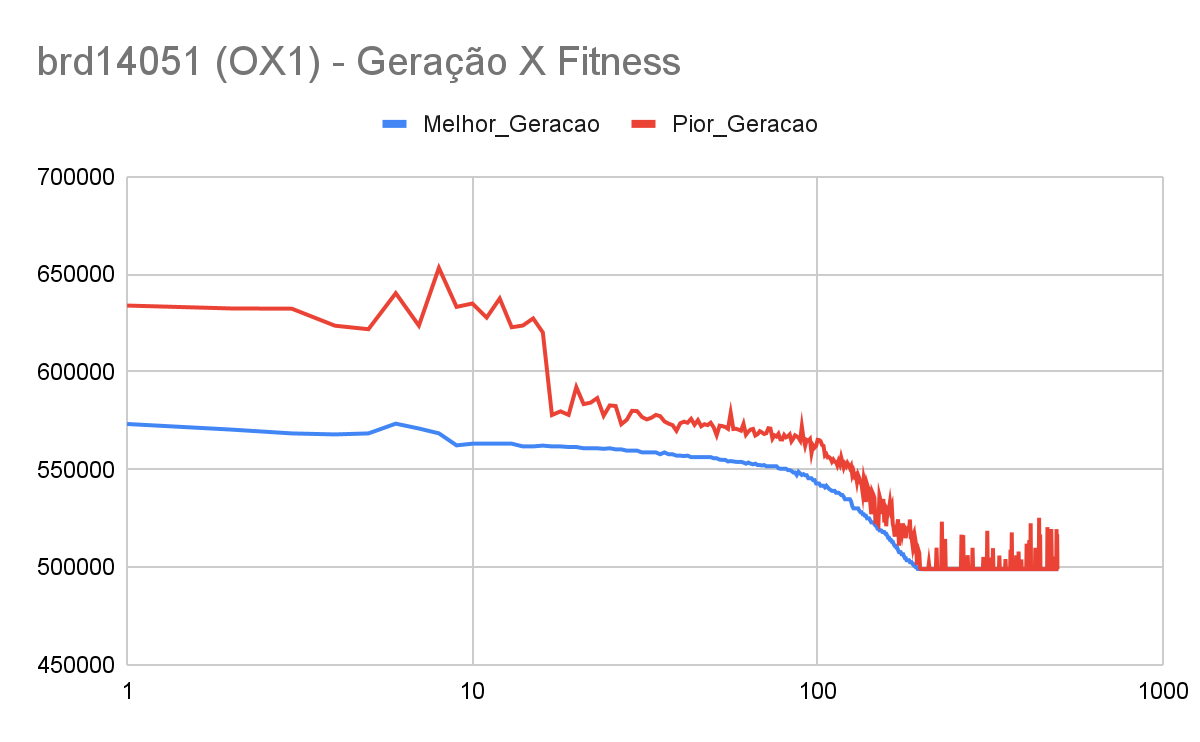


Figura 15 - Gráfico do melhor e pior fitness de cada geração do brd14051- OX1

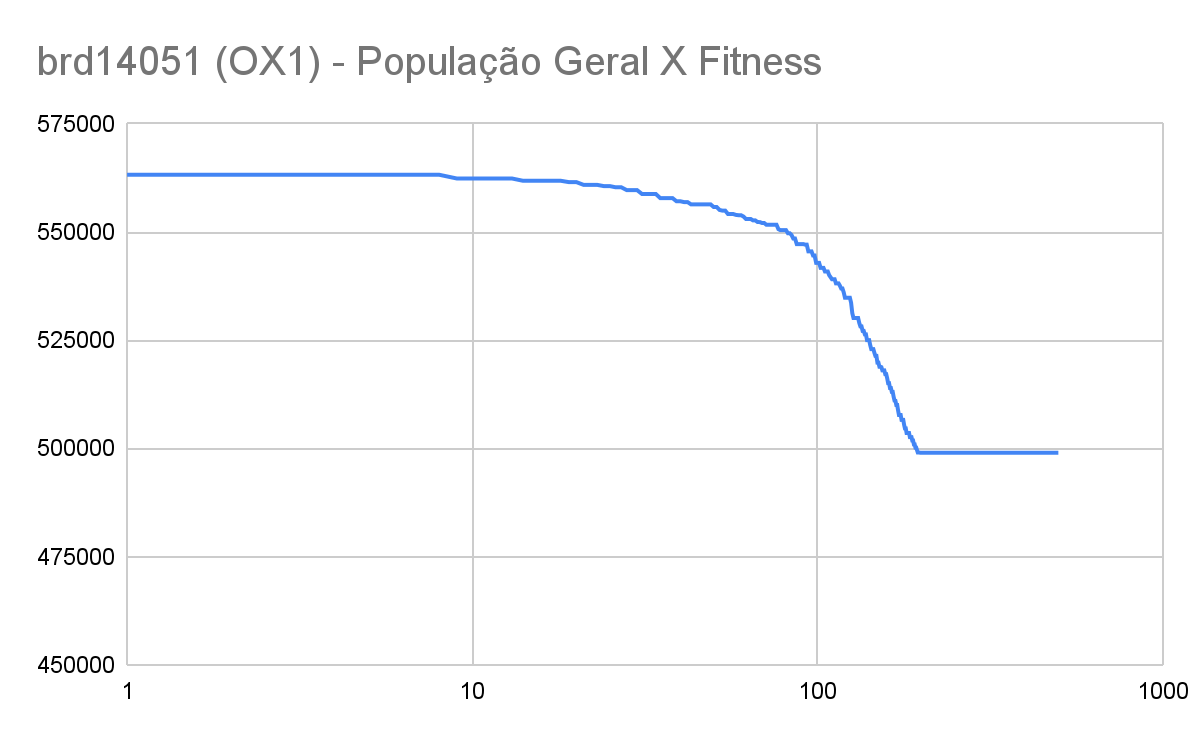


Figura 16 - Gráfico do melhor fitness da população geral do brd14051 - OX1

## 3.5) PCV - d15112.tsp

Tabela 11 - Informações do critério de parada (d15112)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 402 | 9000 | TEMPO\_ESGOTADO |
| **OX2** | 863 | 9000 | TEMPO\_ESGOTADO |

Tabela 12 - Parâmetros utilizados (d15112)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 50 | 50 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 50 | 50 |
| **Taxa de busca local** | 10% | 10% |
| **Taxa de cruzamento** | 80% | 80% |
| **Taxa de mutação** | 5% | 5% |
| **Taxa de sobrevivência** | 60% | 60% |

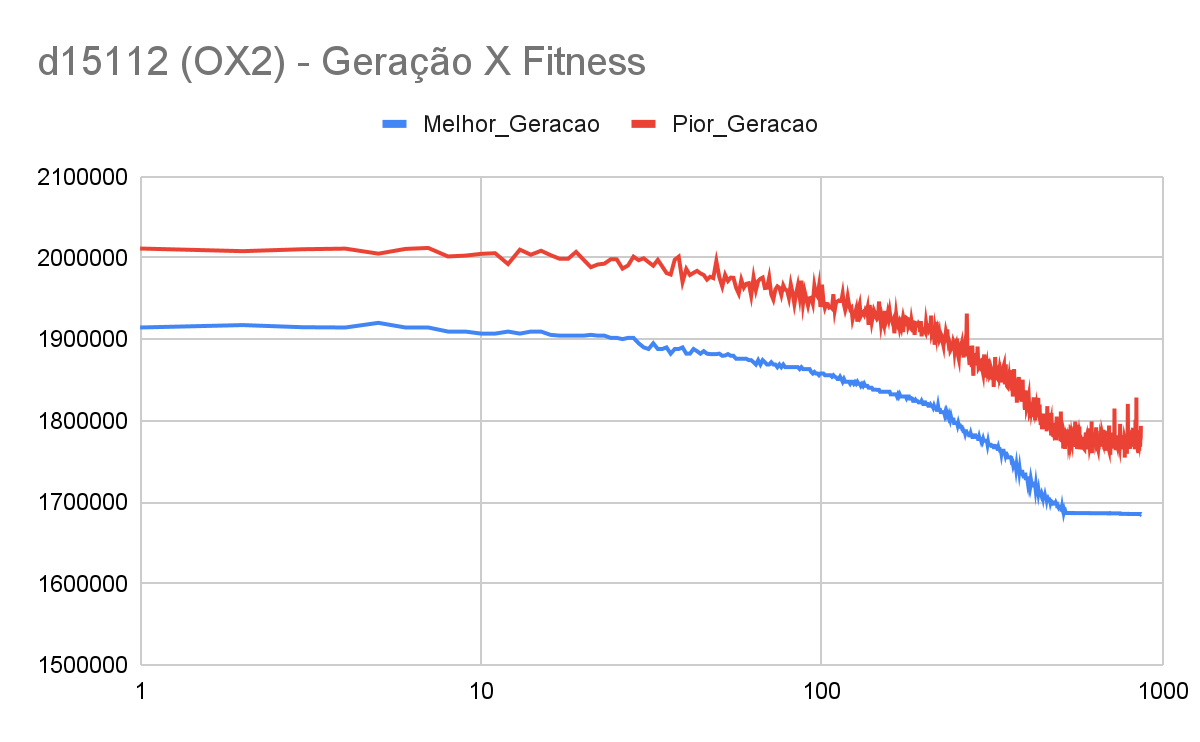
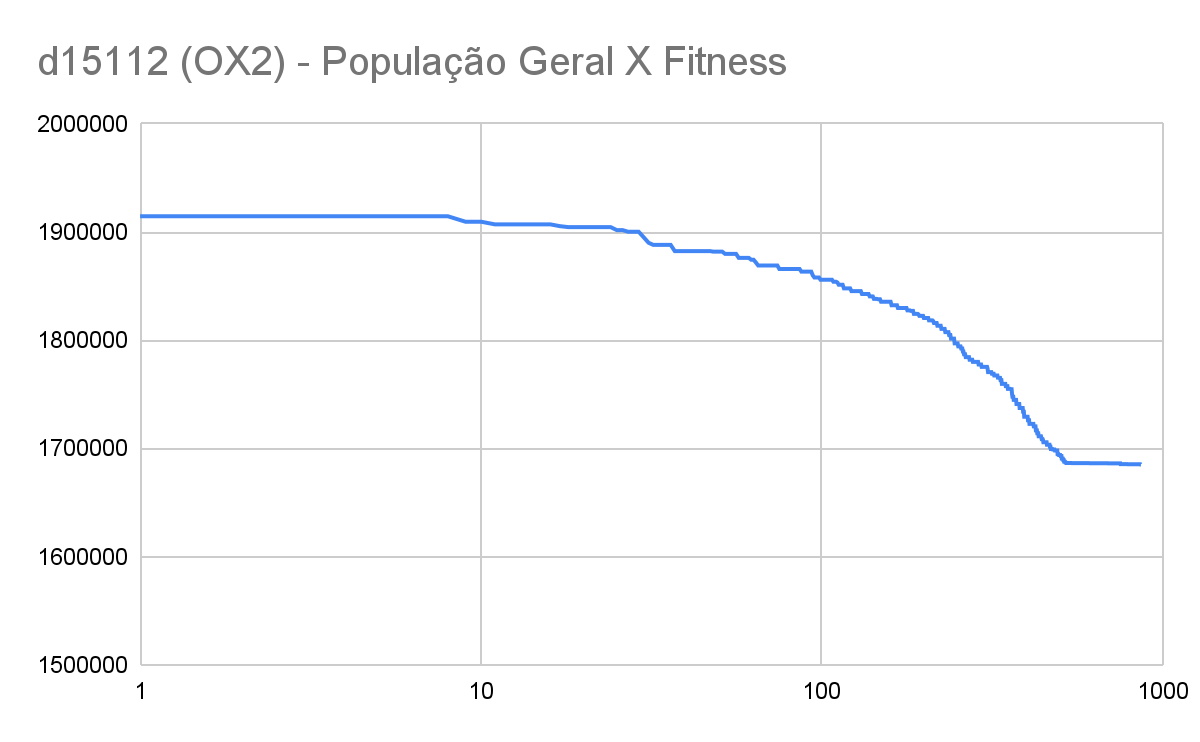


Figura 17 - Gráfico do melhor e pior fitness de cada geração do d15112 - OX2

Figura 18 - Gráfico do melhor fitness da população geral do d15112 - OX2

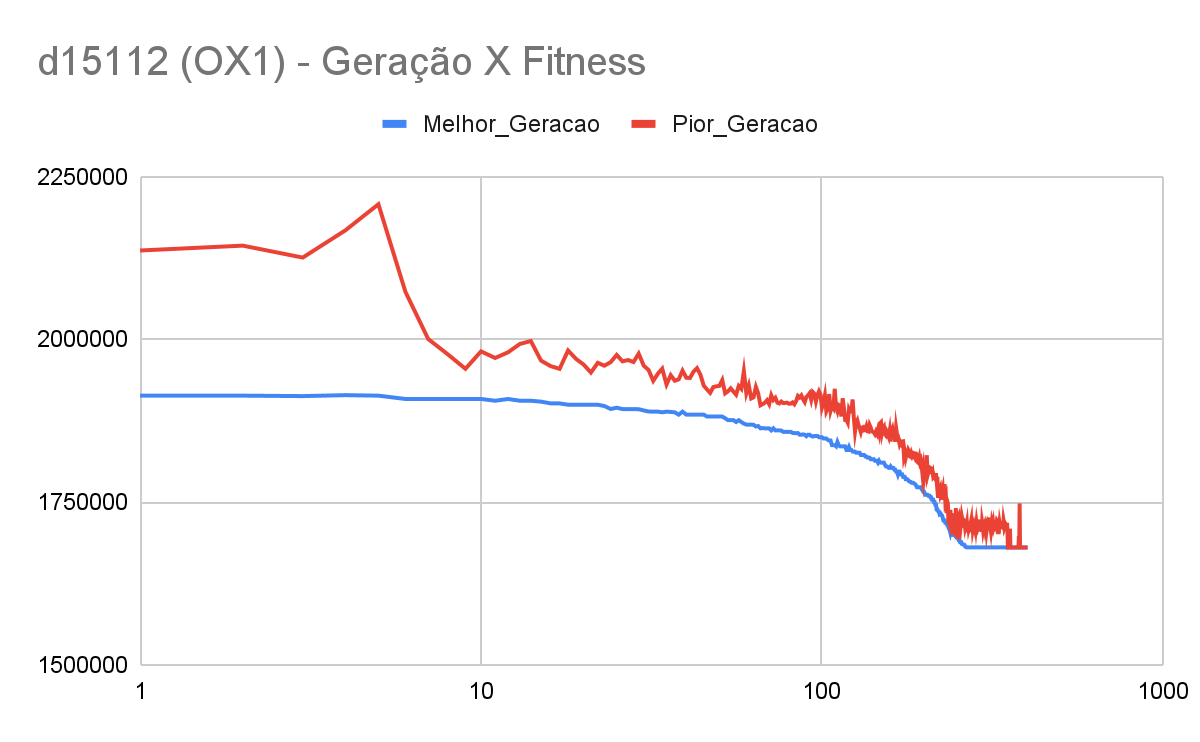


Figura 19 - Gráfico do melhor e pior fitness de cada geração do d15112 - OX1

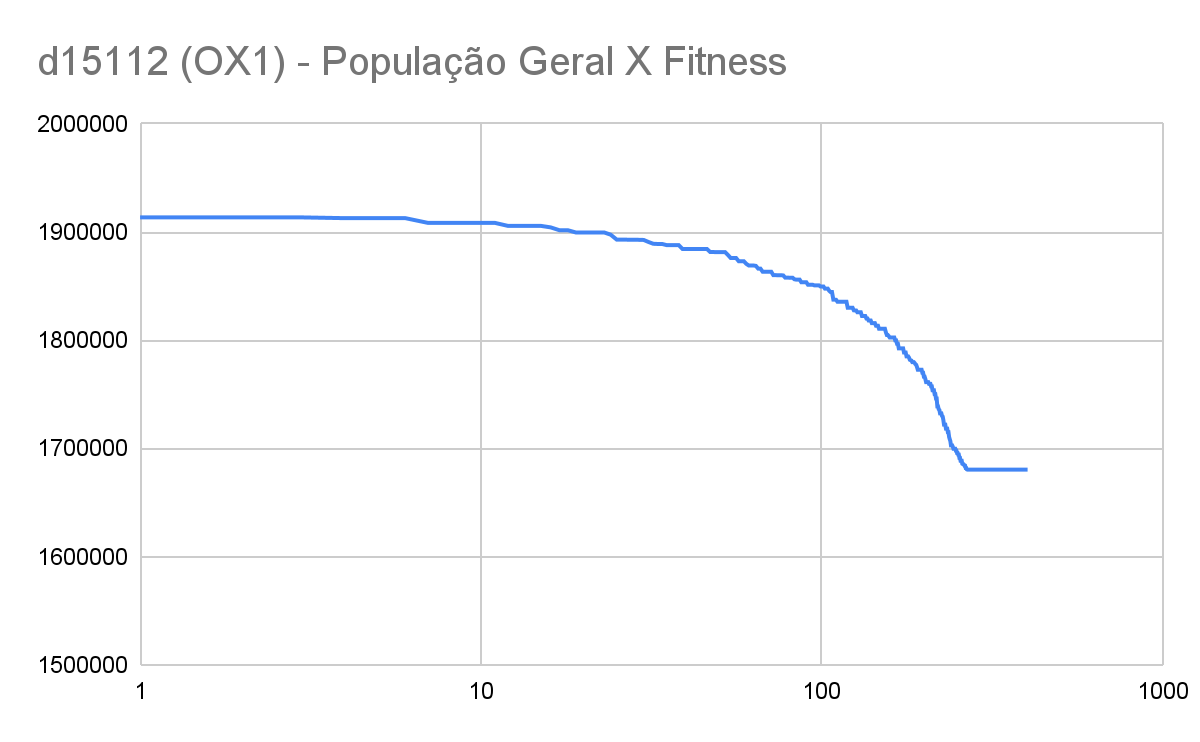


Figura 20 - Gráfico do melhor fitness da população geral do d15112 - OX1

## 3.6) PCV - d18512.tsp

Tabela 13 - Informações do critério de parada (d18512)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 508 | 9000 | TEMPO\_ESGOTADO |
| **OX2** | 878 | 9000 | TEMPO\_ESGOTADO |

Tabela 14 - Parâmetros utilizados (d18512)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 50 | 100 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 50 | 50 |
| **Taxa de busca local** | 10% | 10% |
| **Taxa de cruzamento** | 90% | 90% |
| **Taxa de mutação** | 10% | 5% |
| **Taxa de sobrevivência** | 60% | 40% |

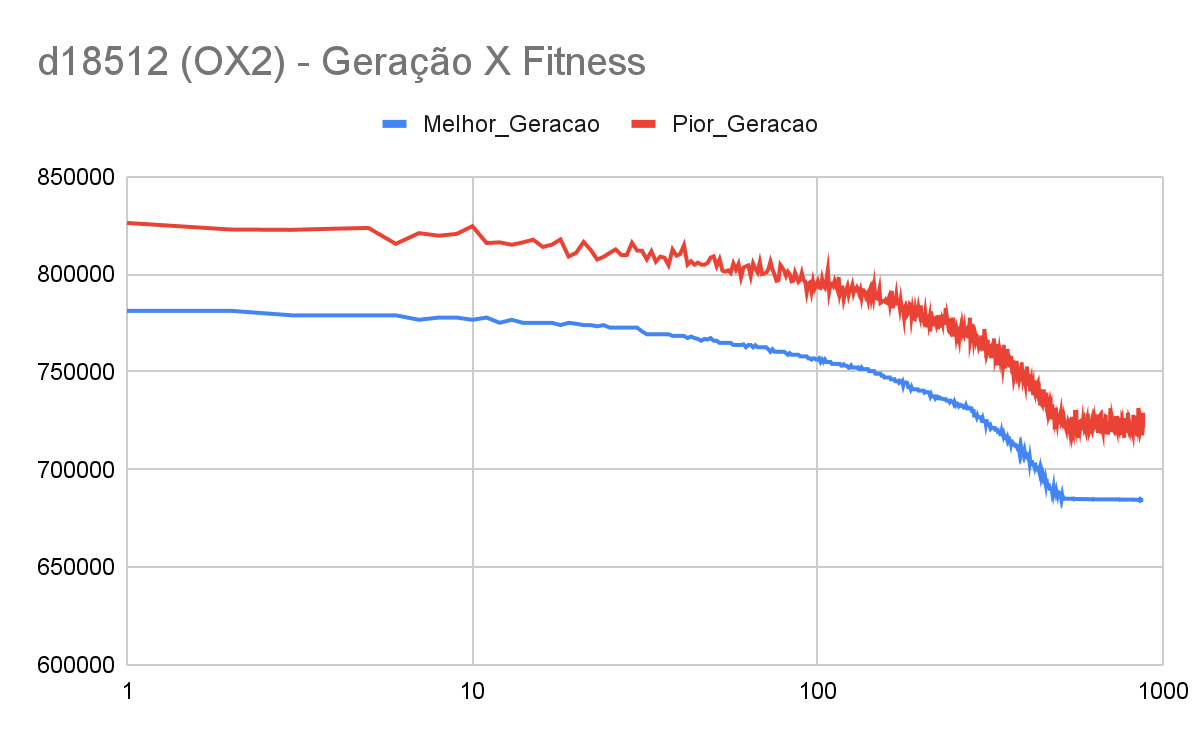


Figura 21 - Gráfico do melhor e pior fitness de cada geração do d18512 - OX2

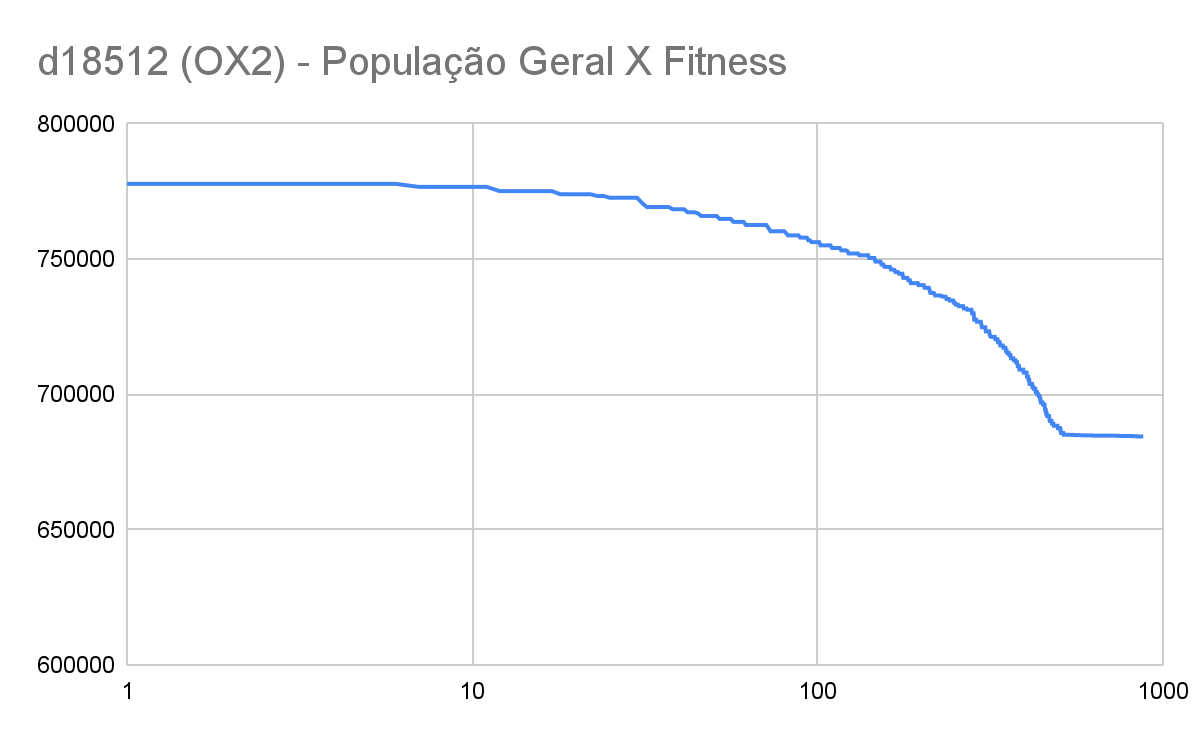


Figura 22 - Gráfico do melhor fitness da população geral do d18512 - OX2

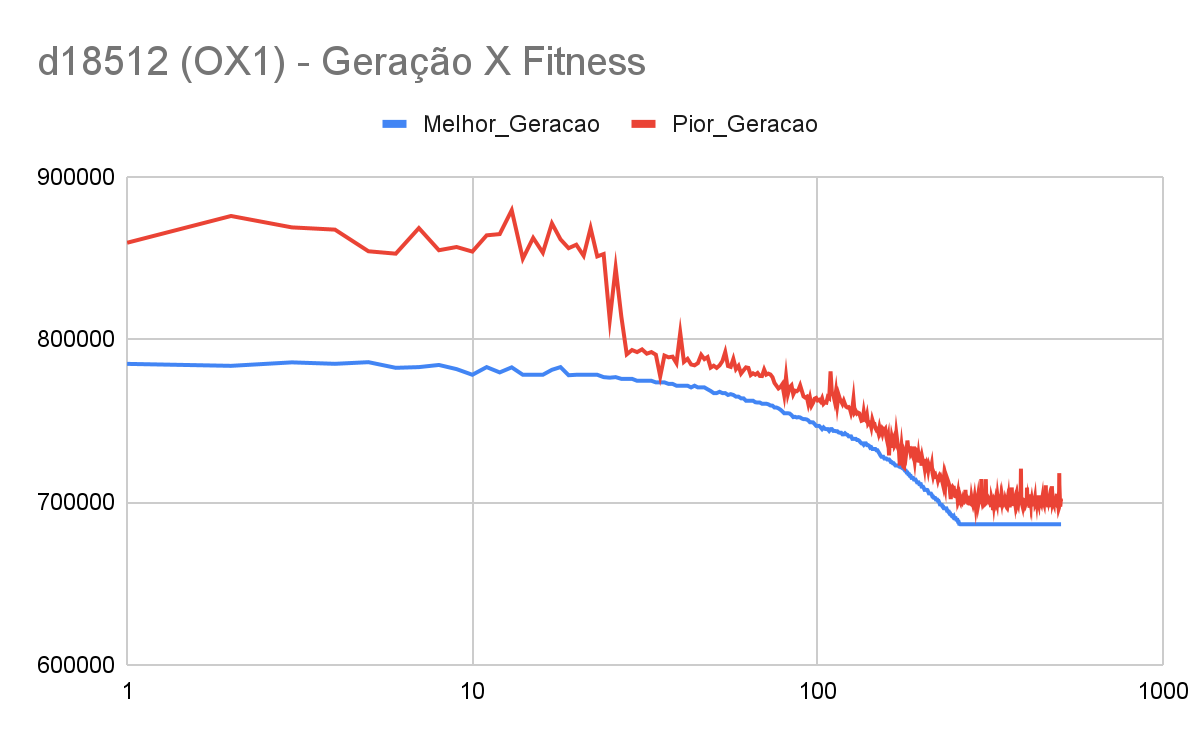


Figura 23 - Gráfico do melhor e pior fitness de cada geração do d18512 - OX1

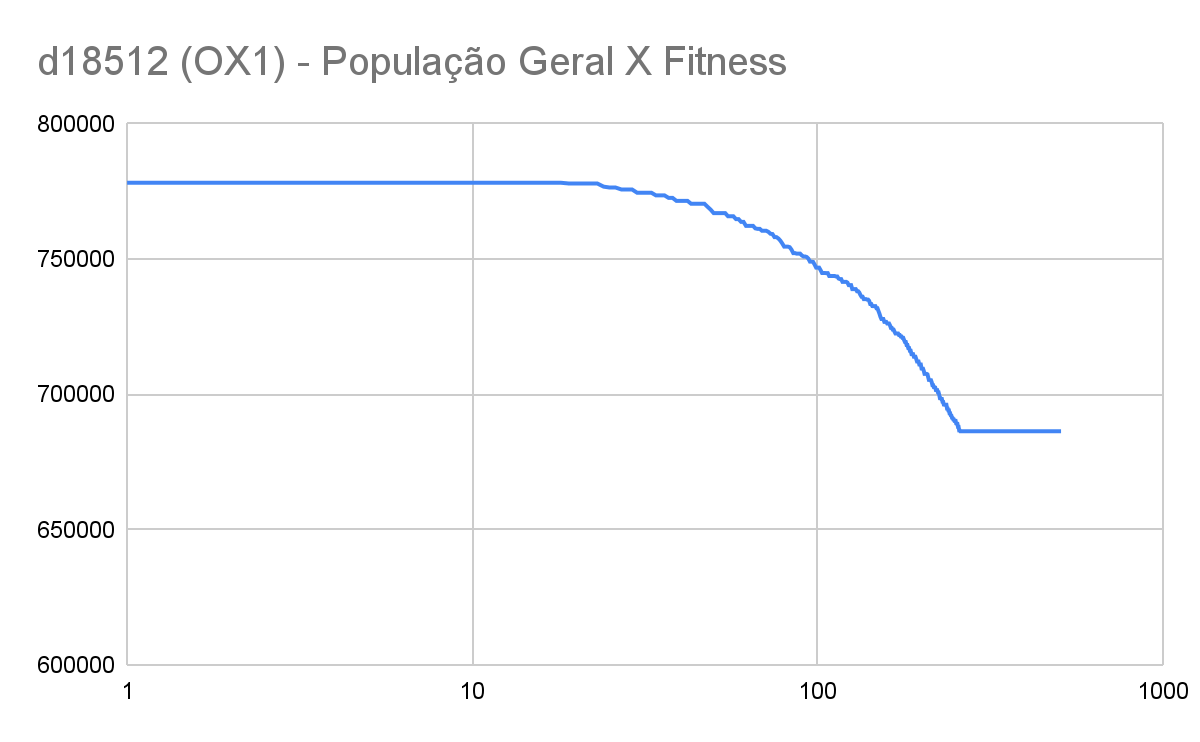


Figura 24 - Gráfico do melhor fitness da população geral do d18512 - OX1

## 3.7) PCV - pla33810.tsp

Tabela 15 - Informações do critério de parada (pla33810)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 118 | 9000 | TEMPO\_ESGOTADO |
| **OX2** | 149 | 9000 | TEMPO\_ESGOTADO |

Tabela 16 - Parâmetros utilizados (pla33810)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 10 | 10 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 200 | 200 |
| **Taxa de busca local** | 40% | 40% |
| **Taxa de cruzamento** | 80% | 80% |
| **Taxa de mutação** | 5% | 5% |
| **Taxa de sobrevivência** | 60% | 60% |

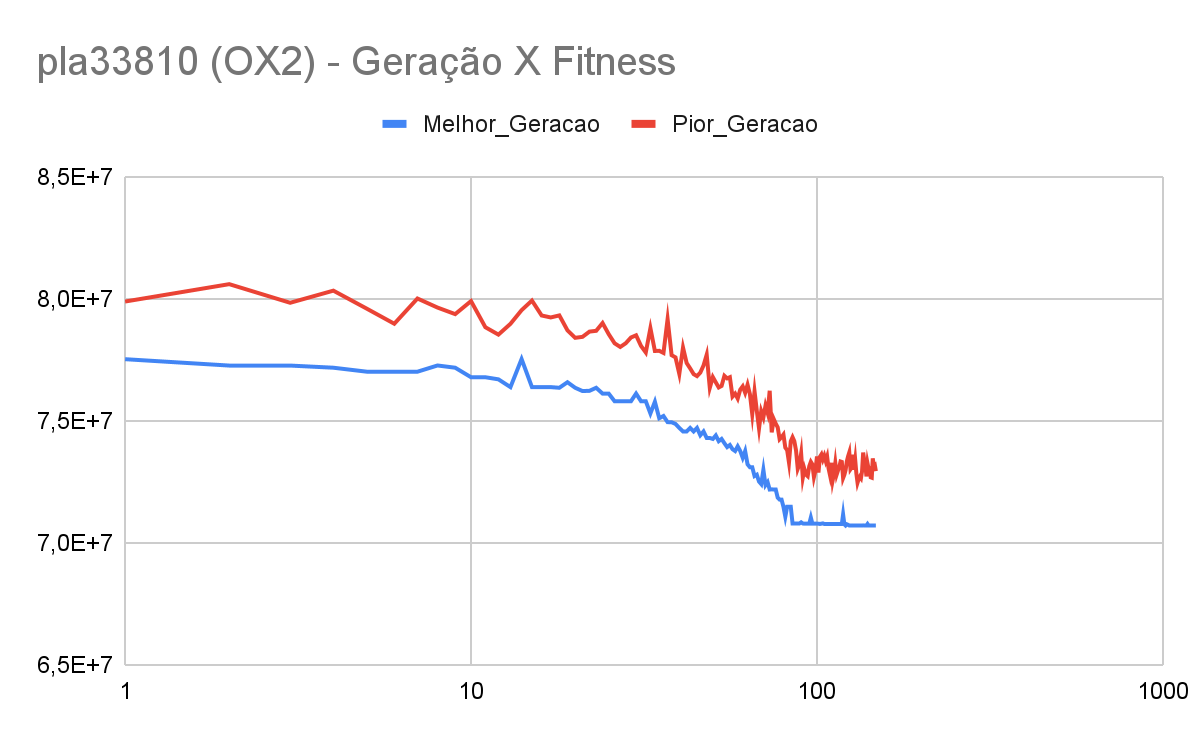


Figura 25 - Gráfico do melhor e pior fitness de cada geração do pla33810 - OX2

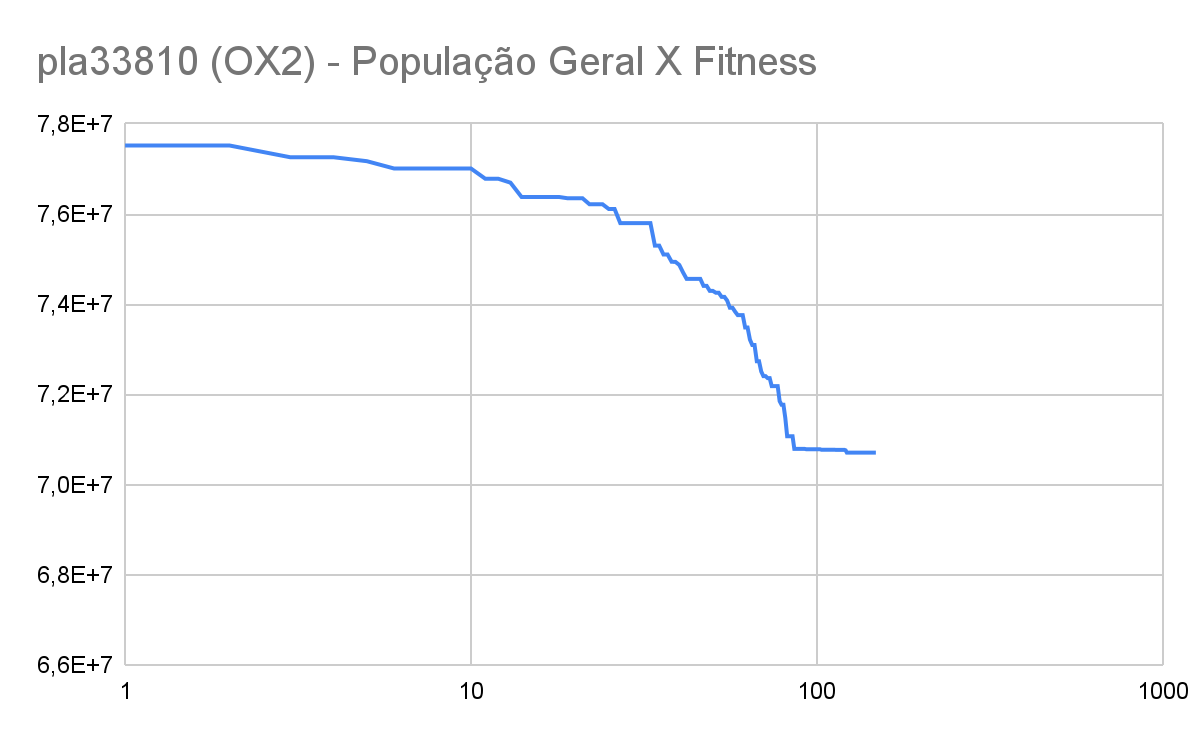


Figura 26 - Gráfico do melhor fitness da população geral do pla33810 - OX2

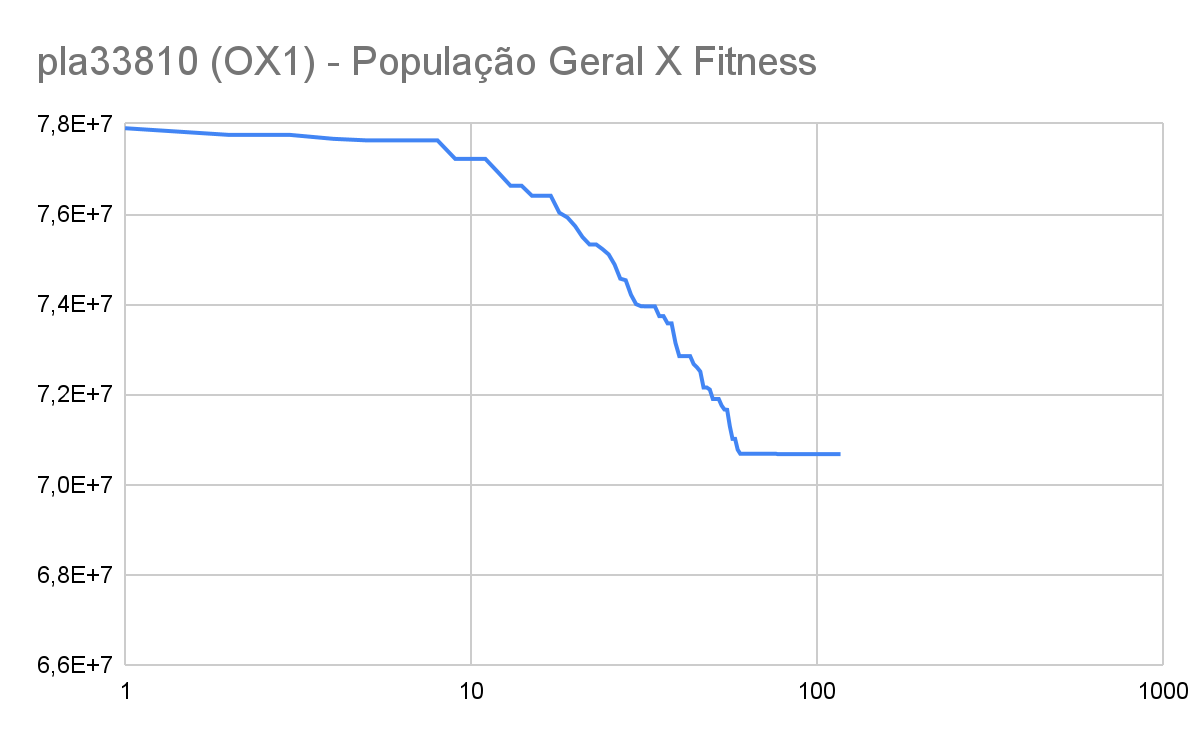


Figura 27 - Gráfico do melhor e pior fitness de cada geração do pla33810 - OX1

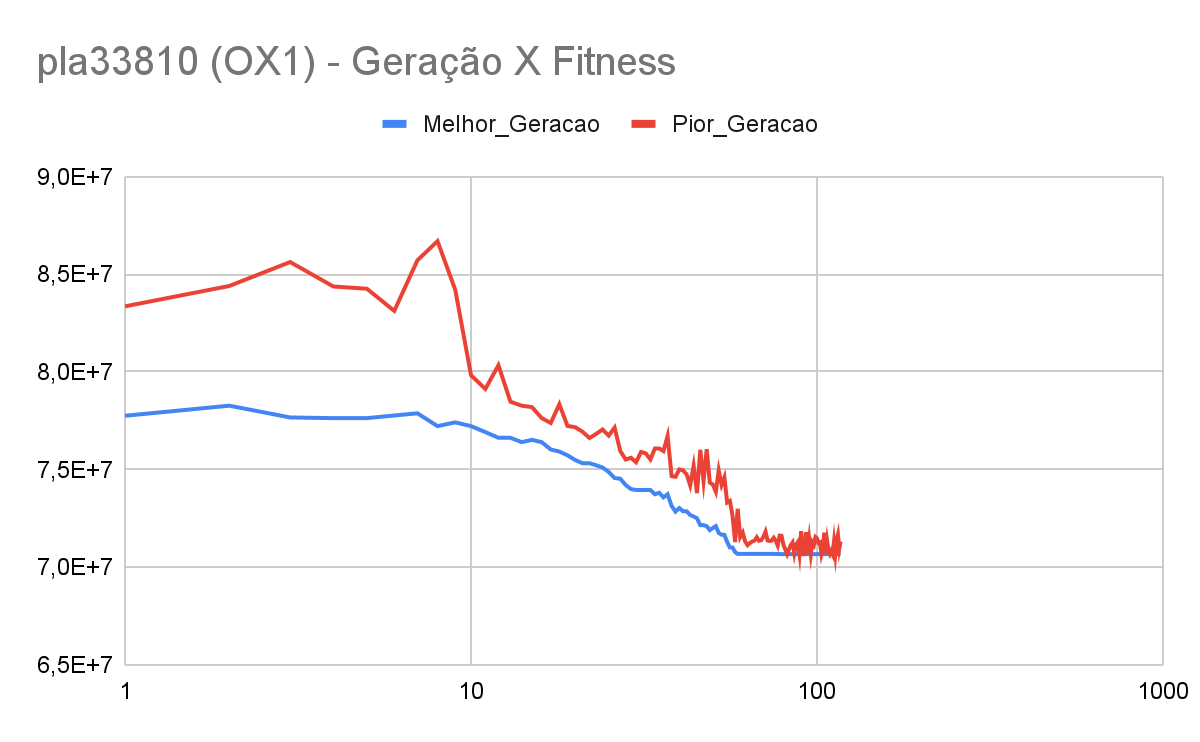


Figura 28 - Gráfico do melhor fitness da população geral do pla33810 - OX1

## 3.8) PCV - pla85900.tsp

Tabela 17 - Informações de critério de parada (pla85900)

| **Algoritmo** | **Geração** | **Tempo de execução (s)** | **Critério de parada** |
| --- | --- | --- | --- |
| **OX1** | 62 | 9000 | TEMPO\_ESGOTADO |
| **OX2** | 173 | 9000 | TEMPO\_ESGOTADO |

Tabela 18 - Parâmetros utilizados (pla85900)

| **Parâmetro** | **OX1** | **OX2** |
| --- | --- | --- |
| **Tamanho da População** | 10 | 10 |
| **Posições de Cruzamento (OX2)** | - | 3 |
| **Profundidade da busca local** | 200 | 200 |
| **Taxa de busca local** | 40% | 40% |
| **Taxa de cruzamento** | 80% | 80% |
| **Taxa de mutação** | 5% | 5% |
| **Taxa de sobrevivência** | 60% | 60% |

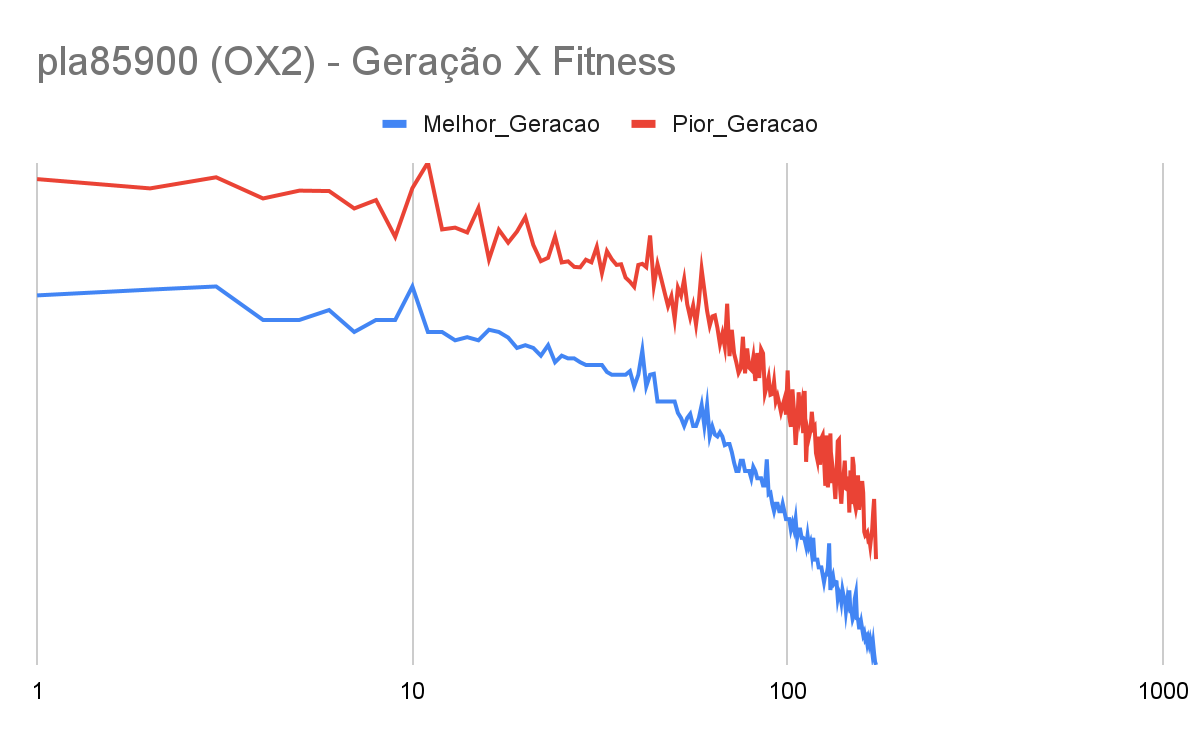


Figura 29 - Gráfico do melhor e pior fitness de cada geração do pla85900 - OX2

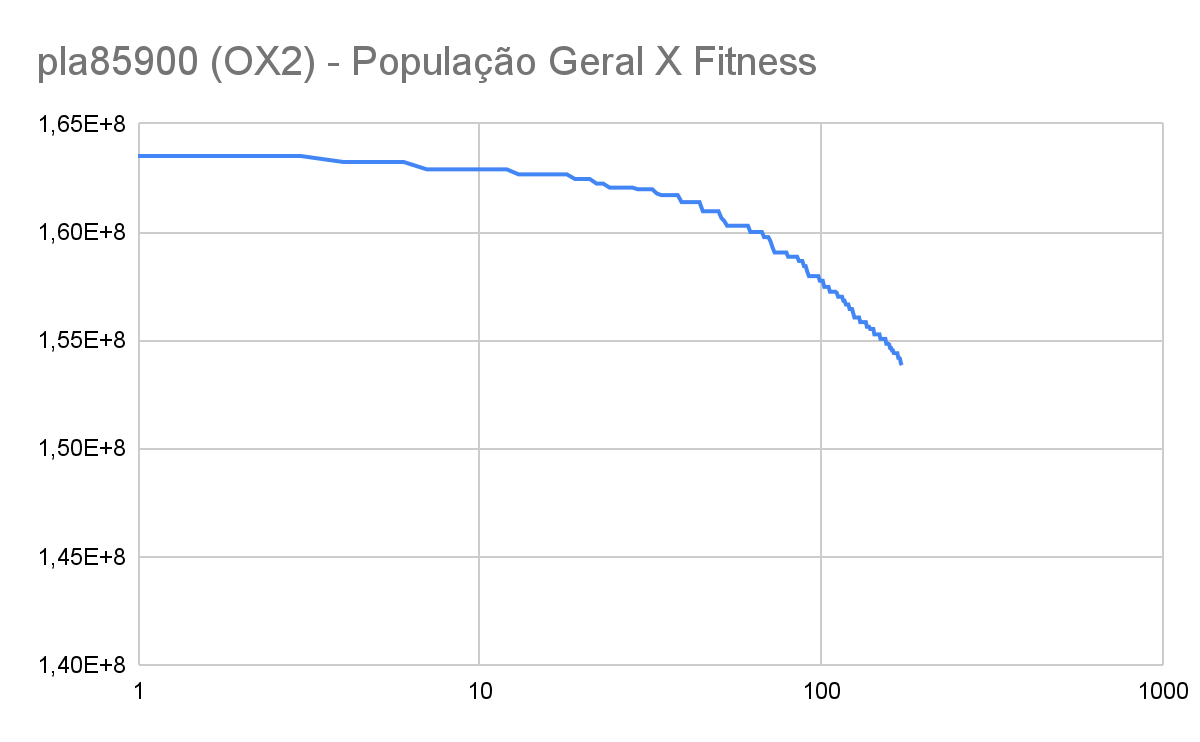


Figura 30 - Gráfico do melhor fitness da população geral do pla85900 - OX2

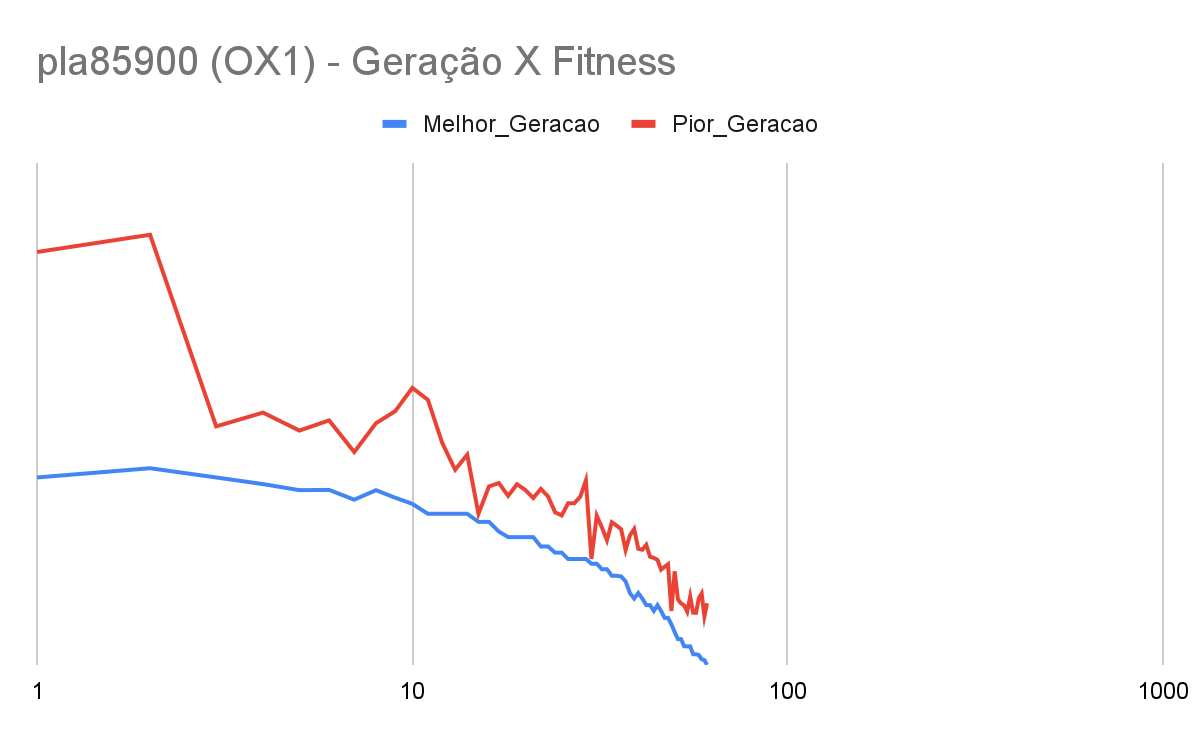


Figura 31 - Gráfico do melhor e pior fitness de cada geração do pla85900 - OX1

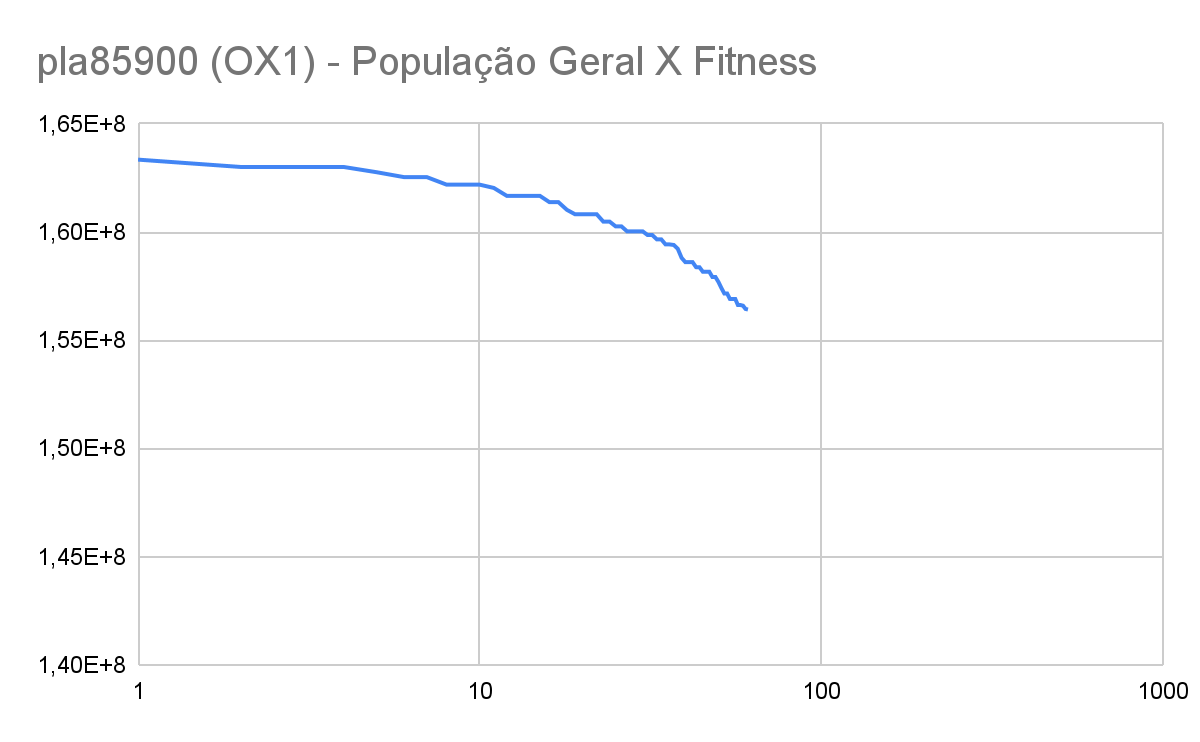


Figura 32 - Gráfico do melhor fitness da população geral do pla85900 - OX1

# 4) Conclusão dos Resultados

Como é possível observar nas tabelas dos resultados obtidos (Tabela 1 e Tabela 2), o uso do algoritmo genético acarretou em uma melhora nos resultados, ou seja, os resultados obtidos através da utilização do algoritmo genético tiveram um GAP menor em relação aos melhores resultados conhecidos na literatura para os casos considerados no trabalho.

Fazendo a análise das tabelas contendo as informações do critério de parada de cada instância, observamos que o algoritmo OX1 sofreu com a estagnação das soluções mesmo para as instâncias menores, além de necessitar um maior tempo de processamento.

A partir da análise dos gráficos do operador OX2 para cada instância, observamos que sempre há uma boa diferença entre o valor do fitness entre o mais apto e o menos apto, ou seja, os filhos gerados pelo operador são bem variados, resultando em uma quantidade maior de gerações para encontrar um resultado satisfatório, porém esta variedade ajuda o algoritmo a não cair em um mínimo local.

De acordo com os gráficos apresentados do operador OX1 para cada instância, é possível observar que ao longo do tempo a diferença entre o valor do fitness do mais apto e do menos apto diminuir, ou seja, ocorre um afunilamento devido à característica do operador de gerar filhos com qualidade maior, acarretando também em uma quantidade menor de gerações para encontrar uma solução boa. Em contrapartida essa característica faz com que o algoritmo fique estagnado em ótimos locais mais rápido.

Considerando as conclusões descritas acima, podemos notar que há uma diferença no modo que são construídos os seus filhos, consequentemente alterando o processo de evolução do algoritmo. Desta forma, comparando os gráficos e tabelas do OX1 e OX2 para cada instância, podemos notar que o operador OX1 encontra um valor satisfatório semelhante ao operador OX2 de forma mais rápida (necessitando de menos geração). Porém como a característica de afunilamento não está presente no OX2, então este ainda possui espaço de melhoria para as próximas gerações.

Levando em consideração os parâmetros considerados e os resultados obtidos a partir dele, é possível observar que o tamanho do problema e da população influenciam bastante no tempo de processamento dos casos, mas também contribuem para uma boa variedade nos indivíduos da população. Outro conjunto de parâmetros que teve bastante impacto na aptidão dos indivíduos em cada geração foi a taxa de busca local e a profundidade da busca local, os quais foram responsáveis pela velocidade em que o algoritmo melhorava as gerações posteriores.